UNIVERSITÀ degli STUDI di BRESCIA

FACOLTÀ di INGEGNERIA

CORSO di LAUREA in INGEGNERIA ELETTRONICA

SPECIALIZZAZIONE in TECNOLOGIE INFORMATICHE

PROGETTO E REALIZZAZIONE DI UNA BASE DI DATI

PER UN SISTEMA ROBOTICO DISTRIBUITO

DOCENTE:
PROF. RICCARDO CASSINIS

STUDENTE:
MARIO PESCATORI

ANNO ACCADEMICO 2010–2011

Indice generale

Capitolo 1 – Introduzione:.....4

1.1 il Sistema Sauron.....4

1.1.1 I Robot di Sorveglianza.....5

1.1.2 AMIRoLoS (Active Marker Internet-based Robot Localization System).....6

1.1.3 APNS (Automatic Predictive Network Selection).....7

1.1.4 ARCSUS (Advanced Robot Control System using SVG).....7

1.2 Funzionamento del Sistema.....8

1.2.1 Server XML-RPC su Morgul.....10

1.2.2 Esempio di Dato da memorizzare nel RDBMS: getStatus.....11

1.3 Pacchetti Software Utilizzati in Sauron.....11

CAPITOLO 2 – Problema.....15

2.1 Cos'è MySQL?.....15

2.1 Gli Storage Engines.....17

2.1.1 Caratteristiche degli Storage Engines InnoDB.....17

2.1.2 Limiti degli Storage Engines InnoDB.....18

2.1.3 Cenni sul Funzionamento degli Storage Engines InnoDB.....19

CAPITOLO 3 – Metodologia di Soluzione.....20

3.1 Strumenti per l'Implementazione del Rdbms.....20

3.1.1 PhpMyAdmin.....20

3.1.2 MySQL Query Browser.....21

3.1.3 MySQL Administrator.....21

3.2 Diagramma delle Classi del RDBMS Sauron.....23

3.3 Stored Procedures e Stored Functions.....24

3.3.1 Funzionamento e Sintassi delle Stored Procedure.....25

3.3.2 Funzionamento e Sintassi delle Stored Function.....26

3.3.3 Funzionamento e Sintassi delle Viste MySQL.....26

3.3.4 I Permessi Necessari alla Creazione delle Viste.....29

CAPITOLO 4 – Descrizione della Soluzione.....30

4.1 Esempi di Chiamate al Server del DB Sauron.....30

4.1.1 La GNU General Public License.....30

4.1.2 NewEventoDB.PHP e TimeStamp.PHP.....31

4.1.3 NewAutoreDB.PHP e LeggiImmagineDB:PHP.....32

4.1.4 PHP LancioRitornoFunzione e ImmaginaDB e Stored Function Invocate.....34

4.1.5 UltimeImmaginiDB.PHP.....35

4.1.6 Altri Esempi di gestione dell'Interfaccia del DB con Sorgente PHP.....36

4.2 Diagramma degli Oggetti del RDBMS Sauron.....39

4.2.1 Tabella Evento.....39

4.3 Procedure e Funzioni di Accesso ai Dati Memorizzate e Gestite Direttamente dal RDBMS.....46

CAPITOLO 5 – Risultati Sperimentali.....48

CAPITOLO 6 – Conclusioni ed Evoluzione.....49

Appendice A – Analisi Dettagliata del Funzionamento delle Stored Procedures.....50

A.1 I permessi necessari.....55

A.2 Le Stored Functions.....56

A.3 Gestire le Stored Procedure e Stored Functions con MySQL Query Browser.....57

Appendice B – Sorgente Photoreporter–V2.1.....59

Bibliografia.....61

CAPITOLO 1 – INTRODUZIONE:

1.1 IL SISTEMA SAURON

Il sistema robotico distribuito SAURON è una rete composta da più periferiche che dialogano ed interagiscono fra loro e l'ambiente in cui cooperano scambiandosi informazioni, comandi e dati. L'acronimo SAURON significa Surveillance Autonomous Robot Over Network (Rete Autonoma di Robot di Sorveglianza) e gestisce tre robot (Morgul, Speedy e Tobor) che autonomamente o pilotati tramite internet svolgono delle perlustrazioni del Laboratorio di Robotica Avanzata dell'Università degli Studi di Brescia. Durante queste perlustrazioni le telecamere ed i sensori montati

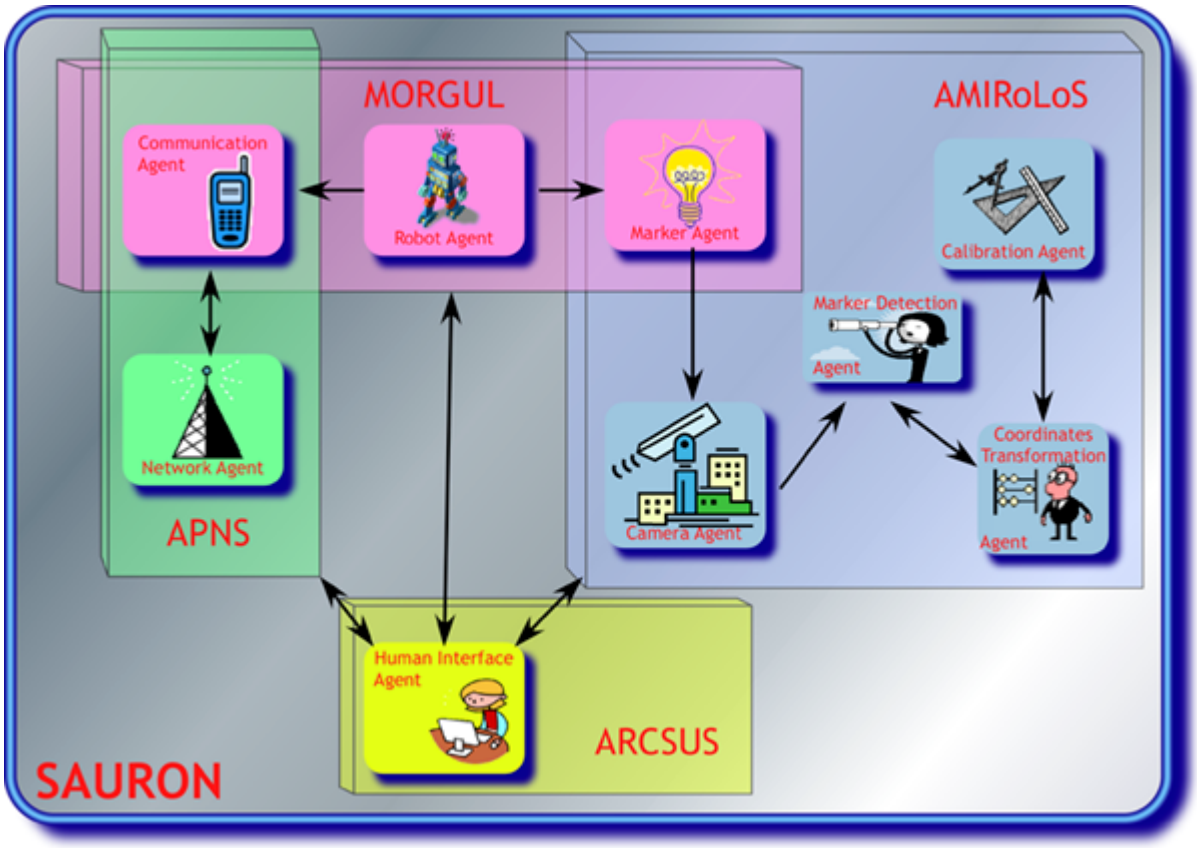


Illustrazione 1: Diagramma a blocchi di SAURON

a bordo rilevano le variazioni dell'ambiente circostante e della loro posizione ed orientamento.

L'architettura globale del sistema è articolata e molto stratificata. Tutte le connessioni tra i vari componenti del sistema sono implementate con connessioni governate dal protocollo TCP/IP ed il sistema operativo montato sui singoli computer della rete è Debian. Nel dettaglio ora presentiamo le quattro componenti del progetto: i robot di sorveglianza, APNS, AMIRoLoS e ARCSUS.

1.1.1 I Robot di Sorveglianza

MORGUL (Mobile Observation Robot for Guarding the University Laboratories)

Il robot utilizzato è Morgul, un robot ActivMedia Pioneer equipaggiato con :

Illustrazione 2: MORGUL lascia la postazione e aziona il marker

- Onboard web-cam
- Active marker
- Laptop (con APNS)
- Librerie di programmazione ARIA (on laptop)
- Dispositivo networking wireless
- Sirena
- Movimento: 4 ruote skid-steer comandate da 4 motori DC ognuno con encoder
- Sonar e bumper anteriori e posteriori per evitare collisioni.

MORGUL è dotato di una stazione di parcheggio e ricarica, che raggiunge utilizzando un semplice sistema ottico che sfrutta le webcam a bordo. La stazione di ricarica permette una autonomia di lunga durata al robot. Morgul raggiunge una velocità massima di 700 mm/s ed è dotato di 16 sonar, 8 davanti e 8 dietro. Il robot è dotato anche del sensore laser UHG-08LX della casa giapponese Hokuyo. Il robot segue un percorso e scansiona l'ambiente in punti predefiniti, memorizzando. L'ambiente di test utilizzato è stato il laboratorio di Robotica Avanzata della Facoltà di Ingegneria.

TOBOR E SPEEDY

Illustrazione 3: Tobor

I robot Tobor e Speedy appartengono alla famiglia dei Pioneer 1m, commercializzata da *ActivMedia Robotics*.

Questi robot utilizzano un sistema di movimento di tipo *differential drive*, con due ruote motrici fisse ed indipendenti ed una ruota folle e pivottante. Dispongono di

sette sonar, posizionati sulla parte anteriore, che consentono di rilevare la presenza di ostacoli fino ad una distanza massima di circa 4 m, spaziando su un angolo di poco più di 180°. Nella parte frontale di Tobor è presente un pinza che consente al robot di afferrare oggetti. Tale dispositivo per la sua conformazione costringe il robot ad entrare nella stazione di ricarica in retromarcia. Infine, sono state installate sulla macchina due webcam, rivolte rispettivamente verso la parte anteriore e posteriore del robot, che consentono di acquisire dati visuali relativi all'ambiente.

Illustrazione 4: Speedy

La comunicazione con i robot avviene attraverso un calcolatore portatile sul quale è installato il pacchetto dei software utilizzati per programmarne la movimentazione, collocato al di sopra degli stessi, e collegato tramite cavo seriale. Sulla parte frontale di Speedy è montata una webcam utilizzata per la discriminazione dei marker attivi.

1.1.2 AMIRoLoS (Active Marker Internet-based Robot Localization System)

Illustrazione 5: Gli active marker di AMIRoLoS

La localizzazione rappresenta uno dei problemi fondamentali nella navigazione dei robot mobili. AMIRoLoS implementa una nuova metodologia volta a posizionare un robot mobile in ambienti chiusi ed aperti utilizzando marcatori attivi e telecamere fuori commercio.

Il sistema di riconoscimento dei marcatori, che è basato sulla differenza tra la frequenza di lavoro dell'otturatore della webcam e del segnale emesso dal marcatore funziona a distanze vicine ai 100 metri in pieno giorno.

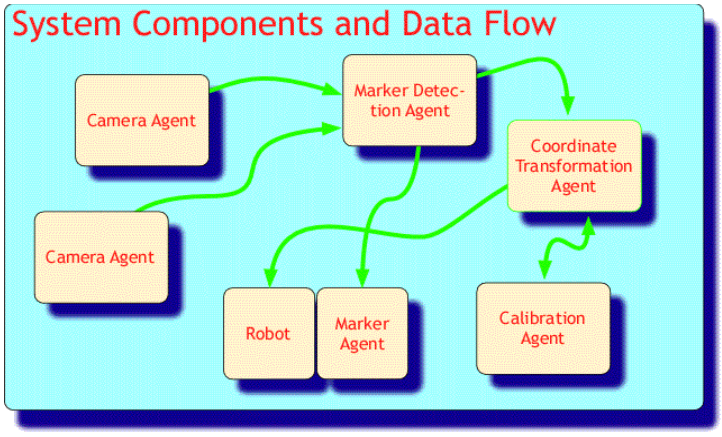


Illustrazione 6: Diagramma a Blocchi di AMIRoLoS

.1.3 APNS (AUTOMATIC PREDICTIVE NETWORK SELECTION)

Illustrazione 7: APNS interfaccia utente

È un pacchetto software progettato per consentire al robot di gestire direttamente l'accesso a reti TCP/IP wireless eterogenee. Il robot è così in grado di cercare un canale di comunicazione disponibile e di stabilire una nuova connessione su di esso. Inoltre se il robot identifica la disponibilità di più reti, può scegliere la migliore in accordo con i criteri predefiniti.

L'obiettivo primario del progetto è di mettere il robot in grado, sempre e ovunque, di utilizzare la migliore via di comunicazione, senza la necessità di doversi disconnettere e riconnettere alle reti disponibili. Questo compito può essere eseguito senza preoccuparsi del contesto in cui il robot si trova. Questo favorisce una flessibilità che permette al robot una immediata operatività senza bisogno di riconfigurare il sistema dopo uno spostamento. Comunque il sistema accumula le precedenti esperienze al fine di anticipare in futuro la scelta migliore.

1.1.4 ARCSUS (ADVANCED ROBOT CONTROL SYSTEM USING SVG)

Illustrazione 8: pannello principale dell'interfaccia AMIRoLoS

Al fine di visualizzare e controllare il sistema, stiamo sviluppando una interfaccia grafica SVG che permetta il controllo del sistema da computer locali o remoti utilizzando i browser standard. L'interfaccia può essere visualizzata su Internet con qualsiasi browser e qualsiasi sistema operativo con un plug-in SVG utilizzando gli standard del W3C (W3C validazione in XHTML 1.0 e CSS).

1.2 Funzionamento del Sistema

Supponiamo che un utente remoto e autorizzato decida di impartire un comando a Morgul: ad esempio il comando di accensione del microcontrollore del robot. L'utente quindi utilizzerà il browser web di un qualsiasi calcolatore collegato ad internet per richiedere la pagina http://frost.ing.unibs.it/protected/fdr/turnOn.php (tipicamente arriverà a richiedere la pagina attraverso un collegamento presente nella pagina principale del sistema di controllo, ma per semplicità si supponga ora che la richiesta si diretta).

Tale richiesta è servita da Frost, calcolatore che assolve, tra l'altro, alla funzione di server web per il laboratorio. La pagina in questione è scritta utilizzando il linguaggio PHP; il server web di Frost pertanto invocherà, al momento di servire la richiesta, l'interprete PHP affinché esegua il codice contenuto nella pagina.

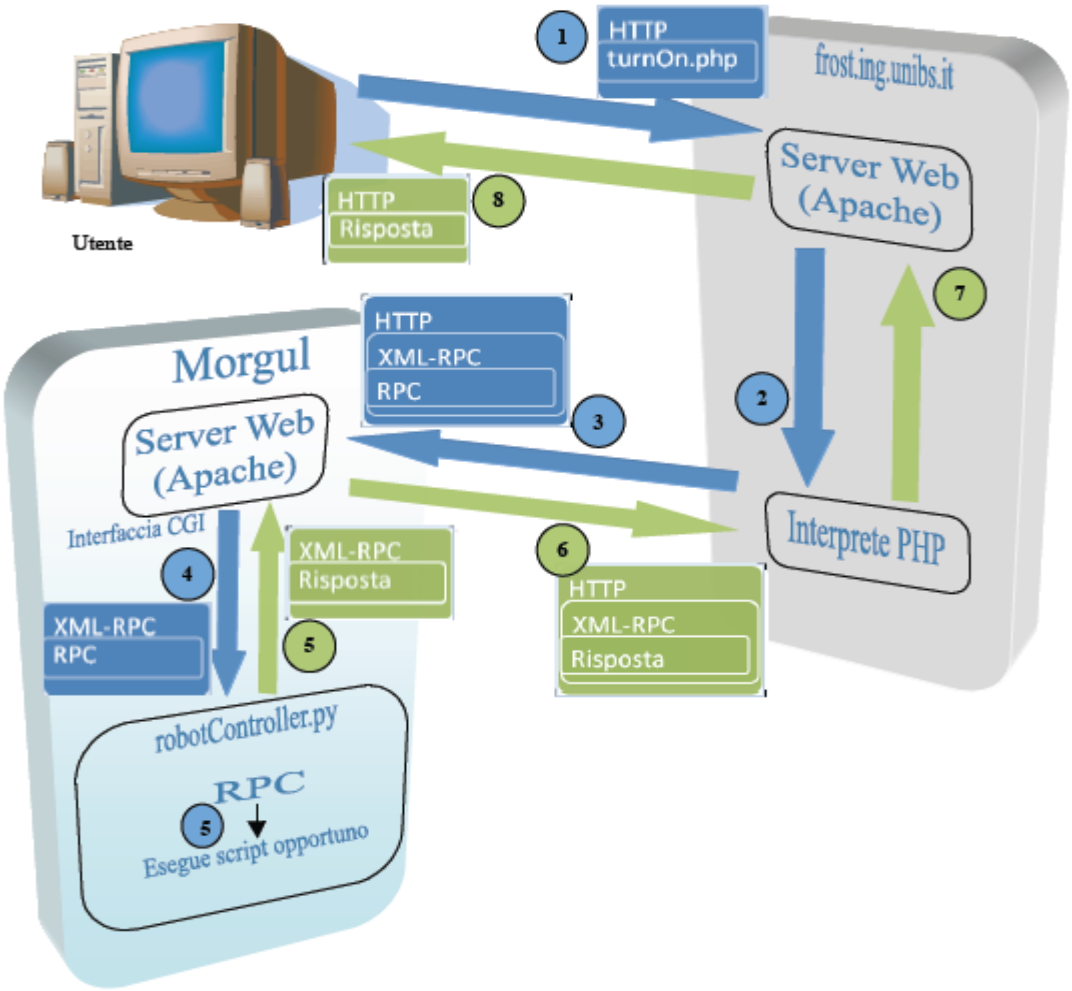


Illustrazione 9: Schema Esecuzione Comando Utente Remoto

Il codice PHP che compone la pagina contiene al suo interno una chiamata a procedura remota (RPC –Remote Procedure Call). Per precisione, si tratta di una chiamata a procedura remota che sfrutta il protocollo XML-RPC e che invoca su Morgul la procedura di nome turnOn.

Poiché lo standard XML-RPC prevede che il trasporto sia effettuato su protocollo HTTP, su Morgul è stato installato il server web Apache. Esso riceve la richiesta HTTP, ne estrae il contenuto XML-RPC (che non saprebbe altrimenti interpretare) e lo inoltra, attraverso l'interfaccia CGI, all'applicazione robotController.

L'applicazione robotController esamina il contenuto della richiesta XML-RPC, estrae il nome della procedura (in questo caso turnOn) e la esegue. La procedura turnOn (in realtà è un metodo, poiché si tratta di un'applicazione ad oggetti) invoca in modo non bloccante lo script di shell onrobot.sh. Essa inoltre restituisce un valore (in questo caso specifico una stringa che contiene il messaggio che il robot è in fase di accensione) il quale viene impacchettato in una risposta XML-RPC e restituito al server web di Morgul.

Il server web di Morgul può a questo punto rispondere alla richiesta XML-RPC che gli era stata inviata dall'interprete PHP in esecuzione su Frost; quest'ultimo a sua volta può usare la risposta ottenuta (che in questo caso contiene la stringa di testo) per confezionare una pagina HTML che restituisce al server web in esecuzione su Frost, il quale, a sua volta, la inoltra al calcolatore dell'utente.

Nel frattempo, poiché la chiamata è non bloccante, su Morgul viene eseguito lo script onrobot.sh. Esso effettua varie operazioni: dispone la riproduzione di un avviso acustico in laboratorio (tramite un'opportuna richiesta CGI verso Frost), registra che il robot è stato acceso creando, su Morgul, il file onrobot e, infine, invoca il programma di gestione del microcontrollore del robot richiedendo l'accensione.

1.2.1 Server XML-RPC su Morgul

Il cuore del sistema è costituito dal server XML-RPC di Morgul. Esso riceve le richieste solo da utenti abilitati per ovvi motivi di sicurezza ed integrità del sistema ed esegue le azioni opportune per soddisfarle, restituendo informazioni circa lo stato del robot. Il protocollo XML-RPC utilizza per il trasporto HTTP, pertanto non presenta particolari problemi nell'interazione con firewall ed altri apparati di rete; le richieste sono espresse in formato XML, garantendo quindi ampia interoperabilità e la possibilità di leggere e comprendere le richieste in caso di necessità di debug.

Inoltre si tratta di uno standard aperto e tutto sommato semplice, ma ciononostante dalle rilevanti potenzialità espressive: permette, ad esempio, il passaggio e la restituzione, oltre che di tipi di dato primitivo, di strutture dati moderatamente complesse quali stringhe, vettori, o array associativi (questi ultimi, chiamati anche dizionari in alcuni linguaggi di programmazione, sono insiemi di coppie chiave-valore).

Non ultimo per importanza, sono disponibili liberamente numerose librerie ed esempi che rendono molto semplice la scrittura tanto di client quanto di server nei più diffusi linguaggi di programmazione, quali C, C++, Python, PHP.

Nel caso specifico, il nostro client è scritto nel linguaggio Python in quanto esso è particolarmente adatto ai compiti di integrazioni e amministrazione di sistemi ed è orientato alla prototipazione rapida.

Esso è costituito dal file /home/sauronweb/public/cgi-bin/robotController.py su Morgul; si tratta infatti di un programma che viene invocato tramite interfaccia CGI dal server web di Morgul al momento dell'arrivo di una richiesta. In realtà le librerie Python per la gestione di XML-RPC sono molto potenti e permettono con sforzo minimo di trasformare robotController.py da applicazione CGI a demone autosufficiente, vale a dire che implementa autonomamente il protocollo HTTP per la parte relativa alla gestione di XML-RPC.

1.2.2 Esempio di Dato da memorizzare nel RDBMS: getStatus

Questa funzione restituisce un insieme di coppie chiave-valore che riassumono lo stato del robot. In particolare, le coppie chiave-valore sono:

chiave	descrizione del valore
robotname	il nome del robot come è restituito dallo stack TCP/IP del sistema operativo
islocked	il valore restituito dalla funzione isLocked
message	il valore restituito dalla funzione getMsg fotografa il valore restituito dalla funzione getPhotoStatus
odometryPosition	il valore restituito dalla funzione getOdometryPosition
robotOn	il valore restituito dalla funzione isRobotOn
stalled	il valore restituito dalla funzione isStalled

Questa funzione restituisce un insieme di coppie chiave-valore che riassumono lo stato del robot ed è il dato più significativo da memorizzare nel RDBMS perché permette di gestire direttamente le movimentazioni e ricostruire le traiettorie percorse. A questo punto è il microprocessore che controlla direttamente il robot che deve, a mio giudizio, invocare il sorgente PHP NewEvento che a titolo esplicativo implementa l'esecuzione della chiamata al DB server per memorizzare il nuovo stato del robot nel paragrafo 4.1.2.

Tutte le funzionalità e i dati del sistema sono distribuiti sui vari calcolatori che compongono la rete e utilizzano vari linguaggi e ambienti di sviluppo, nonché librerie e sistemi client-server che passeremo in rassegna nel prossimo paragrafo.

1.3 Pacchetti Software Utilizzati in Sauron

Il progetto riunisce svariati pacchetti e prodotti che interagiscono e si interfacciano fra loro sfruttando risorse software di vario genere:

- **HTML** è un linguaggio di descrizione per ipertesti. È il linguaggio solitamente usato per i documenti ipertestuali disponibili nel World Wide Web. L'HTML non è un linguaggio di programmazione (in quanto non prevede alcuna definizione di variabili, strutture dati,

funzioni, strutture di controllo) ma solamente un linguaggio di markup che descrive le modalità di impaginazione, formattazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web attraverso tag di formattazione. Tuttavia, l'HTML supporta l'inserimento di script e oggetti esterni quali immagini o filmati.

- **MYSQL** linguaggio di pianificazione del database, ho usato **PHPMYADMIN**
- **server web Apache** con modulo per l'interpretazione codice PHP e programmi CGI
- librerie per il client **XML–RPC**
- applicazioni proprietarie da sorgenti **PHP** e **C/CPP**
- **SHELL BASHB** (acronimo per bourne again shell) è una shell testuale del progetto GNU usata nei sistemi operativi Unix e Unix-like, specialmente in GNU/Linux, ma disponibile anche per sistemi Microsoft Windows. Si tratta di un interprete di comandi che permette all'utente di comunicare col sistema operativo attraverso una serie di funzioni predefinite, o di eseguire programmi. bash è in grado di eseguire i comandi che le vengono passati, utilizzando la redirezione dell'input e dell'output per eseguire più programmi in cascata in una pipeline software, passando l'output del comando precedente come input del comando successivo. Oltre a questo, bash mette a disposizione un semplice linguaggio di scripting che permette di svolgere compiti più complessi, non solo raccogliendo in uno script una serie di comandi, ma anche utilizzando variabili, funzioni e strutture di controllo del flusso.
- **MySQL Query Browser e MYSQLADMIN** analizzate nel paragrafo 3.1.
- **LIBRERIE ARIA** è un pacchetto software scritto in C++ nato come interfaccia per i robot ActiveMedia (MobileRobots). Con Aria è possibile interagire col robot in due modi: inviando semplicemente comandi al robot mediante la classe ArRobot oppure facendo uso delle Action, metodo più sofisticato. Mediante la classe ArNetworking, che sfrutta i protocolli TCP e UDP, è inoltre possibile interagire col robot attraverso la rete. Le classi e i metodi di Aria si fanno dunque carico di compiere le azioni di controllo necessarie a una corretta navigazione del robot: individuare ed evitare gli ostacoli, interagire con i vari sensori (laser, sonar, paraurti) e localizzarsi all'interno di

mappe di ambienti (questo mediante la libreria ausiliaria chiamata Arnl). Il cuore di Aria è costituito quindi dalla classe ArRobot. Essa si occupa di interagire col firmware del robot: riceve tutte le informazioni dai sensori, le elabora e, interagendo con eventuali Action, di cui si parlerà dopo, invia i comandi di movimento al robot. Questo permette una navigazione "intelligente" del robot. Quando si scrive un programma utilizzando Aria, quindi, la prima cosa da fare è stabilire una connessione col robot attraverso un oggetto della classe ArRobotConnector. Esso si occupa anche di elaborare gli eventuali parametri passati dalla linea di comando al momento dell'invocazione del programma. Analogamente, per connettersi ad un laser, si utilizza un oggetto della classe ArLaserConnector. Un oggetto della classe ArRobotConnector fa il suo primo tentativo di connessione con un simulatore (es. MobileSim). Nel caso quest'ultimo non fosse in esecuzione, allora viene stabilita una connessione col robot.

- **Interfaccia CGI** servizio server http sul calcolatore, in maniera tale che ci si potesse connettere via http, visto che la pagina web risiederà su un altro calcolatore, e rendere disponibili in scrittura le porte USB anche per comandi esterni. Il primo problema da affrontare è stato su come implementare il meccanismo che permettesse di far eseguire programmi sul calcolatore montato su Tobor mediante appunto l'interfaccia web. C'erano almeno due possibilità: o attraverso le RPC (remote procedure call), modalità utilizzata per gli altri due robot e utile appunto per far eseguire programmi su un altro calcolatore connesso alla rete, o attraverso le CGI (common gateway interface). Si è optato per le CGI perché più rapide da realizzare. Queste ultime servono in realtà per la creazione di pagine web dinamiche e risultano utili nel caso in cui il server debba fornire all'utente una pagina web al momento, in base a certi parametri ricevuti. Si è però scelto di "aggirare" la vera utilità delle CGI, per ottenere comunque gli stessi risultati che si sarebbero avuti utilizzando le RPC. Facendo uso della shell Bash è possibile creare file con estensione .cgi che all'interno contengono sia il codice html per fare in modo che la CGI sia eseguita, sia comandi veri e propri per far eseguire programmi al server stesso.
- **ARNL** sempre sul sito della MobileRobots è disponibile un software chiamato

"Arnl". In particolare esistono due versioni: una basata sul sonar e l'altra basata invece sul laser. Il compito di questo software è quello di localizzarsi all'interno di un ambiente la cui mappa deve già esistere su file o essere creata al momento. Arnl è un programma "server", ossia un programma la cui utilità sta nel fornire informazioni a un altro eventuale programma detto "client" che si connette ad esso.

- **PYTHON** Python è un linguaggio multi-paradigma, che fa della dinamicità, semplicità e flessibilità i suoi principali obiettivi. Supporta il paradigma object oriented, la programmazione strutturata e molte caratteristiche di programmazione funzionale e riflessione. Le caratteristiche più immediatamente riconoscibili di Python sono le variabili non tipizzate e l'uso dell'indentazione per la definizione dei blocchi. Altre caratteristiche distintive sono l'overloading di operatori e funzioni tramite delegation, la presenza di un ricco assortimento di tipi e funzioni di base e librerie standard, sintassi avanzate quali slicing e list comprehension. Il controllo dei tipi è comunque forte (strong typing) e viene eseguito al runtime (dynamic typing). In altre parole una variabile è un contenitore al quale viene associata un'etichetta (il nome) che può essere associata a diversi contenitori anche di tipo diverso durante il suo tempo di vita. Usa un garbage collector per la liberazione automatica della memoria. Python ha qualche similarità con Perl, ma i suoi progettisti hanno scelto la via di una sintassi più essenziale e uniforme, con l'obiettivo di aumentare la leggibilità del codice. È classificato come linguaggio di scripting, ma pur essendo utile per scrivere script di sistema (in alternativa ad esempio a bash), la grande quantità di librerie disponibili e la facilità con cui questo linguaggio permette di scrivere software modulare favoriscono anche lo sviluppo di applicazioni molto complesse.
- **PHOTOREPORTER V 2.1** file BASH che gestisce l'interfaccia con la webcam , riceve una serie di immagini, una ogni secondo e li memorizza nel database Sauron su Rosie.
- **APPLICAZIONE PROPRIETARIE DI SINGOLI DISPOSITIVI** gestiscono direttamente singoli dispositivi HW e SW che si interfacciano al sistema e potrebbero essere integrati nel DB SAURON.

CAPITOLO 2 – PROBLEMA

Il complesso sistema appena descritto necessita di un database che permetta di memorizzare in modo sicuro ed efficiente gli eventi che ne descrivono l'evoluzione nel tempo, associandoli univocamente al tipo di evento, all'autore che li memorizza ed all'istante in cui l'evento si verifica.

Il database Sauron deve gestire degli eventi asincroni di vario genere provenienti dai processi in evoluzione all'interno del sistema:

- posizione e stato robot
- comunicazioni
- immagini
- allarmi
- timestamp (marca temporale che accerta l'istante del sistema in cui è accaduto l'evento memorizzato)
- registrazioni sonore
- nuovi eventi non ancora classificati
- inizio e termine connessioni

Le chiamate al DB server per memorizzare i dati e leggere tabelle e viste giungono dalla rete invocate da applicativi realizzati in molti dei linguaggi di programmazione elencati in precedenza. Il DBMS deve essere quindi facilmente integrabile e mettere a disposizione le librerie per comunicare con i vari codici garantendo velocità e sicurezza nelle comunicazioni e nell'accesso ai dati.

La scelta si è orientata verso MySQL per le caratteristiche che andiamo a definire.

2.1 COS'È MYSQL?

MySQL, è un Relational database management system (RDBMS), composto da un client e un server che assicurano le seguenti prestazioni:

- Estremamente flessibile
- Ridotti requisiti hardware
- Basso investimento iniziale
- E' la scelta più naturale per gli utilizzatori della pila LAMP (Linux + Apache + MySQL + PHP/Perl/Python)

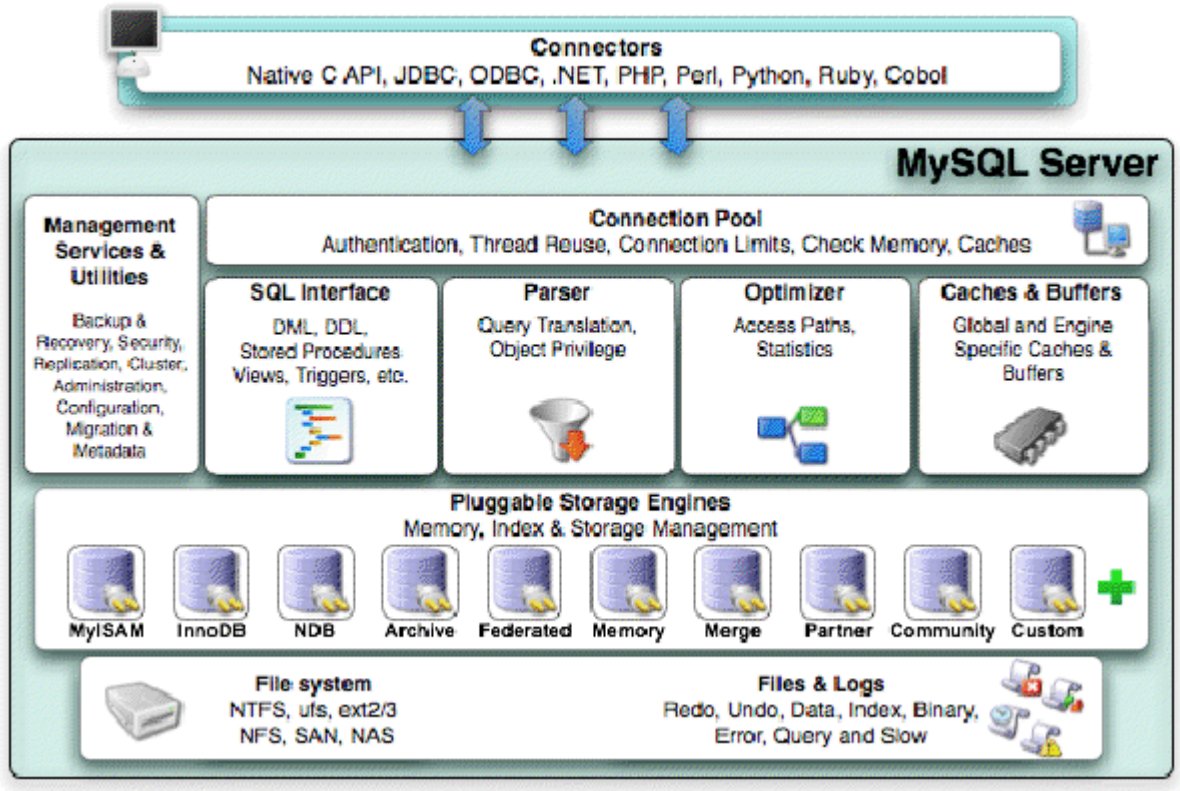


Illustrazione 10: Architettura MySQL

La flessibilità permette a MySQL di essere integrato su diverse piattaforme software:

- Linux
- Windows
- Solaris
- Mac OS X
- FreeBSD
- HP UX
- IBM AIX
- porting da sorgenti

Rende inoltre disponibili connettori e librerie per vari linguaggi di programmazione:

- C/C++ (libmysql)
- Java (JDBC)
- ODBC (Win/Unix)
- PHP
- Perl
- Python
- Altri linguaggi (wrapper libmysql)

2.1 Gli Storage Engines

MySQL memorizza i dati attraverso gli Storage Engines:

- Il motore è legato alla singola tabella
- E' possibile usare motori differenti in una singola query (attenzione...)
- Ad ogni engine è associato uno storage medium
- ogni engine ha delle specifiche features aggiuntive

Per il DBMS Sauron ho scelto lo Storage Engines InnoDB che è sicuramente lo standard attuale e che presenta le maggiori potenzialità anche per sviluppi futuri del sistema.

2.1.1 Caratteristiche degli Storage Engines InnoDB

- Sviluppato da InnoBase (acquisita da Oracle nel 2005)
- Parte integrante di MySQL dalla v4.0
- E' lo storage engine di default dalla v5.5
- Motore transazionale
- Meccanismi avanzati di caching
- Integrità referenziali e foreign key

- MVCC (multiversioning)
- Row level locks
- Crash recovery automatico
- Tablespaces compressi (innodb plugin 1.x)
- Funzionalità del motore InnoDB di MySQL [modifica]
- InnoDB mette a disposizione le seguenti funzionalità:
- transazioni SQL con savepoint e transazioni XA;
- lock a livello di record;
- chiavi esterne;
- integrità referenziale;
- colonne AUTOINCREMENT;
- tablespace.

Il lock in InnoDB per quanto riguarda i comandi SELECT è di tipo non-locking. InnoDB offre delle ottime performance in termini di prestazioni e utilizzo della CPU specialmente quando si ha a che fare con una grande quantità di dati. InnoDB può interagire tranquillamente con tutti gli altri tipi di tabelle in MySQL.

2.1.2 Limiti degli Storage Engines InnoDB

Le tabelle InnoDB sono soggette alle seguenti limitazioni:

- Non è possibile creare più di 1000 colonne per tabella
- Su alcuni sistemi operativi le dimensione del tablespace non possono superare i 2 Gb
- La grandezza di tutti i file di log di InnoDB deve essere inferiore ai 4 Gb
- La grandezza minima di un tablespace è di 10 Mb
- Non possono essere creati indici di tipo FULLTEXT
- Le SELECT COUNT su tabelle di grandi dimensioni possono essere molto lente.

Ritengo che questi limiti siano compatibili con le specifiche del progetto Sauron.

2.1.3 Cenni sul Funzionamento degli Storage Engines InnoDB

Al contrario di quello che fa MYISAM, InnoDB memorizza di default i propri dati all'interno di un unico file definito TABLESPACE. Questo file in base a parametri di configurazione che è possibile specificare e impostare nel file di configurazione di MYSQL può essere AUTO ESTESO (se la dimensione iniziale era per esempio di 50M e li ho riempiti tutti posso dire a MYSQL di aumentare automaticamente di n tot di MB o GB questo file), impostato ad una grandezza voluta, ecc.

È Possibile anche usare più TABLESPACE per memorizzare i dati con InnoDB (In questo caso l'unico file che può essere auto esteso sarà solo l'ultimo). È molto comodo utilizzare più file per memorizzare i dati quando si ha a che fare con sistemi operativi che non supportano file troppo grandi (alcuni sistemi operativi max 2 Gb), in questo modo si aggira il problema.

Vediamo quindi una serie di funzionalità avanzate garantite dalle tabelle INNODB:

- Foreign Key Le tabelle INNODB sono in grado di gestire l'integrità referenziale tra le chiave esterne del database, in sostanza tale integrità è sempre stata delegata all'abilità dello sviluppatore, ora con il motore INNODB è possibile delegare questa annosa attività a MySQL stesso, potendo quindi specificare vari comportamenti a seconda della chiave esterna utilizzata.
- Transazioni Le transazioni sono fondamentali in situazioni tipiche da shop online dove sino a che non arriva conferma da parte della banca o di chi valida la carta di credito, le query non devono essere "eseguite realmente".Fondamentalmente si tratta di eseguire una serie di query che verranno validate o annullate mediante chiamate COMMIT o ROLLBACK.
- Mancanza – FULLTEXT questa è una mancanza grave nelle tabelle INNODB, l'assenza di poter dichiarare un campo come indice fulltext per poter eseguire ricerche con score alla Google.

Attualmente non interessa direttamente il progetto, potrebbe condizionare la creazione di interfacce web per visualizzare tabelle, viste e quindi uno storico degli eventi del sistema.

CAPITOLO 3 – METODOLOGIA DI SOLUZIONE

Si è deciso di installare il RDBMS MySQL su un singolo calcolatore denominato Rosie dedicato esclusivamente come server del Database Sauron. I vincoli richiesti in termini di prestazioni e capacità di memorizzazione non evidenziavano criticità particolari e quindi il calcolatore non necessitava di particolari requisiti. Rosie è andata in crash un paio d'anni fa e il database non è più stato ricaricato su altri server anche a seguito di un elaborato di altri studenti che hanno riprogettato il DB Sauron nel 2011. Attualmente rimane solo la presente documentazione del progetto che ha perso il suo carattere sperimentale.

3.1 STRUMENTI PER L'IMPLEMENTAZIONE DEL RDBMS

Per creare la struttura del DB, scrivere procedure, funzioni e viste, verificare i dati inseriti, le query per le interrogazioni e monitorare le prestazioni ho utilizzato principalmente phpmyadmin, mysqladministrator e mysqlquerybroser.

3.1.1 PHPMYADMIN

PhpMyAdmin è uno strumento software gratuito scritto in PHP destinato a gestire l'amministrazione di MySQL sul World Wide Web. phpMyAdmin supporta una vasta gamma di operazioni con MySQL. Le operazioni più frequentemente utilizzate sono supportate dall'interfaccia utente (gestione di basi di dati, tabelle, campi, relazioni, indici, utenti, permessi, ecc), mentre hai ancora la possibilità di eseguire direttamente qualsiasi istruzione SQL.

Supporto per la maggior parte delle caratteristiche di MySQL:

- sfogliare e drop database, tabelle, viste, campi e indici
- creare, copiare, eliminare, rinominare e modificare database, tabelle, campi e indici
- manutenzione del server, database e tabelle, con proposte di configurazione del server
- eseguire, modificare e bookmark alcuna SQL-statement, anche batch-query
- gestire gli utenti MySQL e privilegi

- gestire le stored procedure e trigger
- Importazione di dati da CSV e SQL
- Esportare i dati in vari formati: CSV, SQL, XML, PDF, ISO / IEC 26300, OpenDocument Text e foglio di calcolo, Word, LaTeX e altri
- Amministrazione di più server
- Creazione grafica PDF del layout del database
- Creazione di query complesse utilizzando query-by-example (QBE)
- Ricerca a livello mondiale in un database o un sottoinsieme di esso
- Trasformare i dati archiviati in qualsiasi formato utilizzando una serie di funzioni predefinite, come visualizzare BLOB-data come immagine o un download-link

3.1.2 MySQL Query Browser

è lo strumento che ho utilizzato per interrogare il database e verificarne il funzionamento. MySQL Query Browser è uno strumento grafico fornito da MySQL AB per creare, eseguire ed ottimizzare query in un ambiente grafico. È progettato per permettere, di effettuare query ad un server MySQL ed analizzare i dati memorizzati nel database.

Se da una parte tutte le query eseguite in MySQL Query Browser possono essere effettuate anche tramite l'utility da riga di comando mysql, MySQL Query Browser consente di eseguire query e modificare dati in maniera più intuitiva, tramite un'interfaccia grafica. MySQL Query Browser è progettato per funzionare con le versioni di MySQL 4.0 e superiori.

3.1.3 MySQL Administrator

MySQL Administrator è un programma per effettuare operazioni di amministrazione, come ad esempio la configurazione, il monitoraggio e l'avvio e l'arresto di un server MySQL, gestione degli utenti e connessioni, eseguire backup, e una serie di altre attività amministrative. La maggior parte delle attività possono essere eseguite utilizzando un client a riga di comando, come mysqladmin o mysql, ma MySQL Administrator offre i seguenti vantaggi:

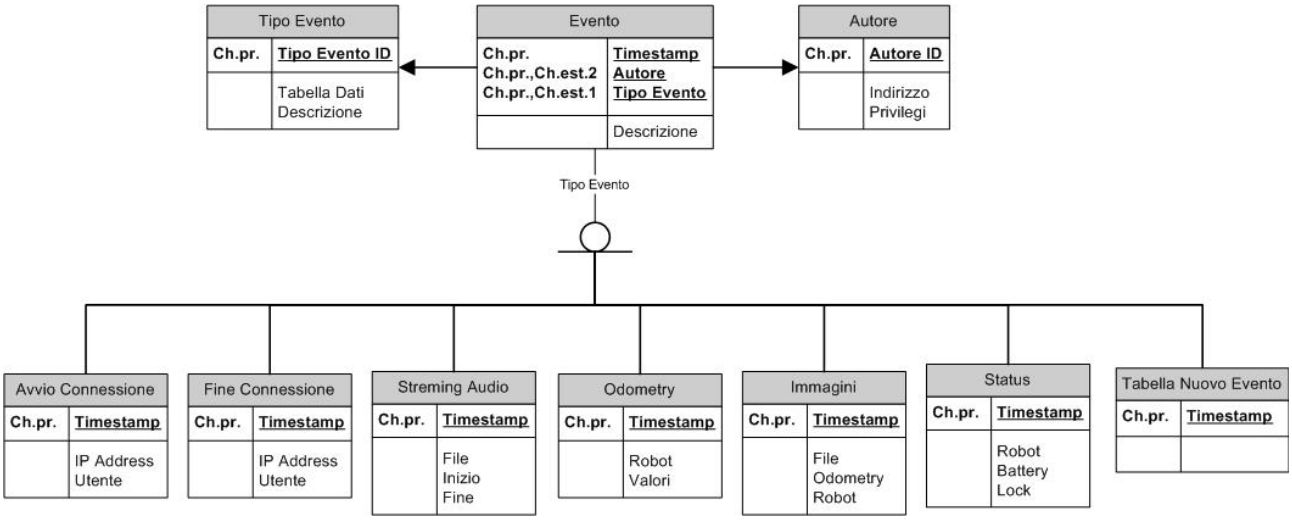
- La sua interfaccia grafica lo rende più intuitivo.
- Esso fornisce una migliore visione d'insieme delle impostazioni che sono cruciali per le prestazioni, l'affidabilità e la sicurezza dei vostri server MySQL.
- Mostra degli indicatori grafici di performance, rendendo così più facile determinare e regolare le impostazioni del server.
- MySQL Administrator è progettato per funzionare con le versioni di MySQL 4.0 e superiori.
- Interfaccia web intuitiva

Supporta la maggior parte delle caratteristiche di MySQL:

- sfogliare e drop database, tabelle, viste, campi e indici
- creare, copiare, eliminare, rinominare e modificare database, tabelle, campi e indici
- manutenzione del server, database e tabelle, con proposte di configurazione del server
- eseguire, modificare e bookmark alcuna SQL-statement, anche batch-query
- gestire gli utenti MySQL e privilegi
- gestire le stored procedure e trigger
- Importazione di dati da CSV e SQL
- Esportare i dati in vari formati: CSV, SQL, XML, PDF, ISO / IEC 26300 - OpenDocument Text e foglio di calcolo, Word, LaTeX e altri
- Amministrazione di più server
- Creazione grafica PDF del layout del database
- Creazione di query complesse utilizzando query-by-example (QBE)
- Ricerca a livello mondiale in un database o un sottoinsieme di esso
- Trasformare i dati archiviati in qualsiasi formato utilizzando una serie di funzioni predefinite, come visualizzare BLOB-data come immagine o un download-link

3.2 Diagramma delle Classi del RDBMS Sauron

Ecco di seguito il progetto del database Sauron che memorizza gli eventi asincroni che giungono dalla rete del Laboratorio di Robotica Avanzata dell'Università.




```
-- phpMyAdmin SQL Dump
-- version 2.9.1.1-Debian-6
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generato il: 01 Feb, 2008 at 03:25 PM
-- Versione MySQL: 5.0.32
-- Versione PHP: 5.2.0-8+etch9
--
-- Sauron Project
--
--
-- Database: `Sauron`
--
```

Illustrazione 11: Schema Tabelle Data Base Sauron

Ogni evento ha come chiave primaria il tempo di sistema nel quale è stato notificato al RDBMS che quindi svolge la funzione di orologio unico del sistema per sincronizzare gli autori che gli trasmettono l'evento che hanno generato direttamente o registrato dalle loro periferiche. Ogni evento ha una sua classe di appartenenza che lo delinea nelle sue caratteristiche e rende il sistema flessibile e facilmente aggiornabile in base alle sopravvenute esigenze di sviluppo. Le due chiavi esterne mettono appunto in relazione l'evento con il suo autore e la classe dei tipi di evento. Le altre classi rappresentano i vari tipi di eventi ammessi e si differenziano fra loro per i dati significativi per la loro memorizzazione.

3.3 Stored Procedures e Stored Functions

Passiamo ora ad illustrare come i vari autori possono interagire con il RDBMS SAURON.

La scelta è caduta su una funzionalità disponibile a partire dalla versione MySQL 5.0, le Stored Procedure e le Stored Function, che sono routine particolarmente utile in determinate situazioni:

- quando le applicazioni client sono scritte in diverse lingue o devono lavorare su piattaforme diverse, ma hanno bisogno per eseguire le operazioni dell'unico database.
- quando la sicurezza è fondamentale. Le banche, per esempio, utilizzare le stored procedure e funzioni per tutte le operazioni più comuni. Ciò fornisce un ambiente coerente e sicuro, e le routine in grado di garantire che ogni operazione sia correttamente registrata. In una tale impostazione, le applicazioni e gli utenti non hanno accesso alle tabelle del database direttamente, ma possono solo eseguire specifiche routine memorizzate.
- forniscono prestazioni migliori perché meno informazioni devono essere inviate tra il server e il client. Il compromesso è che questo fa aumentare il carico sul server di database perché più del lavoro è fatto sul lato server e meno è fatto sul lato client (applicazione). Questo va preso in considerazione in casi come il nostro dove molte macchine client (come i server Web) sono serviti da un solo server di database.
- consentono inoltre di avere le librerie di funzioni del server database. Questa è una caratteristica condivisa dai linguaggi di applicazioni moderne che consentono tale disegno interno (ad esempio, utilizzando le classi). Utilizzando queste funzionalità l'applicazione, scritta nella lingua del client, è benefica anche per il programmatore, anche al di fuori del campo di utilizzo del database.

3.3.1 Funzionamento e Sintassi delle Stored Procedure

Una **stored procedure** (o "procedura memorizzata") è un insieme di istruzioni SQL che vengono memorizzate nel server con un nome che le identifica; tale nome consente in seguito di rieseguire l'insieme di istruzioni facendo semplicemente riferimento ad esso. Vediamo una semplice stored procedure di esempio:

```
    CREATE PROCEDURE nome_procedura (IN param1 INT, OUT param2 INT)
SELECT COUNT(*) INTO param2 FROM tabella
WHERE campo1 = param1;
```

La procedura è attribuita ad uno specifico database, esattamente come una tabella. Di conseguenza la procedura viene assegnata ad un database al momento della creazione. In fase di creazione, quindi, se indichiamo il nome della procedura senza specificare il database questa sarà assegnata al db attualmente in uso. Ogni procedura può avere uno o più parametri, ciascuno dei quali è formato da un nome, un tipo di dato e l'indicazione se trattasi di parametro di input, di output o entrambi. Se manca l'indicazione, il parametro è considerato di input.

Il codice di sopra crea una procedura chiamata 'nome_procedura' nel database in uso; la procedura usa un parametro in input e uno in output, entrambi interi, ed effettua il conteggio delle righe in tabella in cui il valore di campo1 corrisponde al primo parametro; il risultato della query viene memorizzato nel secondo parametro attraverso la clausola INTO. Per richiamare la procedura e visualizzare il risultato, dovete scrivere il seguente codice:

```
    CALL nome_procedura (5, @a);
SELECT @a;
```

Con l'istruzione CALL effettuiamo la chiamata della procedura, passando il valore 5 come parametro di input e la variabile @a come parametro di output, nel quale verrà memorizzato il risultato. La SELECT successiva visualizza il valore di tale variabile dopo l'esecuzione. Nell'esempio appena visto la stored procedure conteneva una semplice SELECT; è possibile invece creare procedure che contengono sintassi complesse comprendenti più istruzioni: in pratica, dei veri e propri script, con la possibilità di controllare il flusso attraverso vari costrutti (IF, CASE, LOOP, WHILE, REPEAT, LEAVE, ITERATE).

3.3.2 Funzionamento e Sintassi delle Stored Function

Le **stored functions** sono simili alle stored procedures, ma hanno uno scopo più semplice, cioè quello di definire funzioni e devono sempre ritornare un valore. Esse restituiscono un valore, e non possono quindi restituire resultset, al contrario delle stored procedures. Nelle versioni di MySQL precedenti alla 5.0 esistevano le "user-defined functions", che venivano memorizzate esternamente al server. Ora queste funzioni sono ancora supportate, ma è sicuramente consigliabile utilizzare le nuove stored functions. Vediamo una semplice stored function di esempio:

```
    CREATE FUNCTION func (n1 INT, n2 INT, n3 INT, n4 INT)
RETURNS INT
DETERMINISTIC
BEGIN
DECLARE avg INT;
SET avg = (n1+n2+n3*2+n4*4)/8;
RETURN avg;
END
```

Questa funziona ritorna la media pesata di 4 numeri interi; i termini DECLARE e DETERMINISTIC servono rispettivamente per dichiarare una variabile e per specificare a MySQL che fornendo lo stesso input, il risultato non cambia.

3.3.3 Funzionamento e Sintassi delle Viste MySQL

Le viste sono comunemente considerate un modo per mostrare i dati di un database con una struttura diversa da quella che hanno effettivamente sulla base dati.

Un uso possibile delle viste è quello di concedere ad un utente l'accesso ad una tabella mostrandogli solo alcune colonne: tali colonne saranno inserite nella vista, sulla quale l'utente avrà i permessi di accesso, mentre gli saranno negati quelli sulla tabella sottostante.

Altre possibili applicazioni riguardano la possibilità di leggere dati da più tabelle contemporaneamente attraverso JOIN o UNION, oppure di comprendere dati che non sono fisicamente presenti sulla tabella in quanto calcolati a partire da altri dati.

Le viste non possono avere lo stesso nome di una tabella facente parte dello stesso database. Ecco la sintassi da usare per la creazione di una vista:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
```

```
[DEFINER = { utente | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nome [(lista_colonne)]
AS istruzione_select
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

La clausola **OR REPLACE** consente di sostituire una vista con lo stesso nome eventualmente già esistente. La clausola **SQL SECURITY** stabilisce se, al momento dell'esecuzione, la procedura utilizzerà i permessi dell'utente che la sta eseguendo o quelli dell'utente che l'ha creata (il default è **DEFINER**).

La lista delle colonne, opzionale, viene usata per specificare i nomi delle colonne dalle quali è composta la vista. È possibile ometterla e adottare come nomi di colonne quelli restituiti dalla SELECT.

L'istruzione **SELECT** è quella che definisce quali dati sono contenuti nella vista. Essa può essere anche molto complessa, fare riferimento a più tabelle, contenere subquery, ma è sottoposta ad alcuni limiti: non può utilizzare variabili, non può contenere subquery nella FROM, non può fare riferimento a tabelle temporanee ma solo a tabelle già esistenti. Non possono esistere viste temporanee.

Una caratteristica essenziale delle viste è quella di essere o meno **aggiornabili**: una vista aggiornabile infatti consente di modificare i dati della tabella sottostante, cosa che non è possibile in caso contrario. Tuttavia una vista, per essere aggiornabile, deve soddisfare determinate condizioni: in pratica, deve esistere una relazione uno a uno fra le righe della vista e quelle della tabella sottostante.

Quindi la SELECT che genera la vista non può contenere funzioni aggregate, DISTINCT, GROUP BY, HAVING, UNION, subquery nella lista delle colonne selezionate, JOIN (con qualche eccezione, vedere oltre); inoltre non può avere un'altra vista non aggiornabile nella clausola FROM, o una subquery nella WHERE che fa riferimento a una tabella nella FROM.

Le viste possono essere generate con due algoritmi: MERGE o TEMPTABLE. Con

quest'ultimo, la SELECT relativa alla vista viene utilizzata per creare una tabella temporanea, sulla quale viene poi eseguita l'istruzione richiesta; con MERGE invece la SELECT viene "mescolata" con l'istruzione che richiama la vista stessa. Una vista non è mai aggiornabile se creata con TEMPTABLE.

La clausola **ALGORITHM** che abbiamo visto nella CREATE può essere usata per specificare quale algoritmo usare: tuttavia per poter usare MERGE è necessario che la vista soddisfi alcuni dei criteri che abbiamo visto in precedenza per le viste aggiornabili. Se utilizziamo TEMPTABLE per una vista che potrebbe essere costruita con MERGE, avremo la conseguenza di non poterla usare per aggiornamenti, ma il vantaggio di una maggior velocità nel rilascio della tabella sottostante. Se non indichiamo la clausola ALGORITHM, il default sarà UNDEFINED.

Quando una vista è aggiornabile possiamo anche utilizzarla per **inserire** dati, purché contenga tutte le colonne della tabella che non hanno un valore di default, e nessuna colonna derivata da espressioni.

In alcuni casi, come eccezione a quanto detto prima, può essere aggiornabile una vista che contiene una INNER JOIN, a patto che gli aggiornamenti vengano fatti sui campi di una sola tabella. Sulla stessa vista è possibile fare delle INSERT, anche in questo caso su una sola tabella.

La clausola **WITH CHECK OPTION** si usa nei casi di viste aggiornabili, per impedire inserimenti o aggiornamenti di righe della tabella che non soddisfano la condizione prevista nella clausola WHERE della vista. Qualora la vista faccia riferimento ad altre viste, l'ulteriore specificazione di **CASCADED** fa sì che il controllo venga propagato alle condizioni delle viste interne; se invece viene specificato **LOCAL** il controllo viene effettuato solo al primo livello, cioè sulla vista attuale. Il default è CASCADED.

Per modificare una vista si utilizza una istruzione ALTER che è praticamente identica alla CREATE OR REPLACE:

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
```

```
[DEFINER = { utente | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nome [(lista_colonne)]
AS istruzione_select
[WITH [CASCADED | LOCAL] CHECK OPTION]
L'eliminazione della vista si ottiene con l'istruzione DROP VIEW:

DROP VIEW [IF EXISTS] nome
```

Infine è possibile visualizzare la definizione della vista attraverso la seguente istruzione:

```
SHOW CREATE VIEW nome
```

3.3.4 I Permessi Necessari alla Creazione delle Viste

Per creare una vista occorre il privilegio CREATE VIEW (nonché il privilegio SUPER se si vuole definire un altro utente come DEFINER); per eliminarla il privilegio DROP. Entrambi i privilegi sono necessari per modificare una vista (sia con ALTER VIEW che con CREATE OR REPLACE).

Infine il privilegio SHOW_VIEW consente di visualizzare la definizione della vista. Ovviamente sono poi necessari i permessi richiesti per accedere ai dati selezionati; tali permessi sono richiesti al creatore o all'esecutore della vista a seconda del parametro SQL SECURITY.

CAPITOLO 4 – DESCRIZIONE DELLA SOLUZIONE

Per connettere la rete del laboratorio ARL al RDBMS Sauron funzionante su Rosie ho scritto dei codici sorgenti in PHP che invocano a titolo esemplificativo le nuove istanze tramite stored function e procedures e poi analizzano i dati storici utilizzando le viste.

4.1 ESEMPI DI CHIAMATE AL SERVER DEL DB SAURON

In questo paragrafo i sorgenti PHP che chiamano alcune funzioni e procedure memorizzate all'interno del motore del DB Sauron seguite dal codice SQL che ne descrive il funzionamento. Il progetto del RDBMS Sauron è stato realizzato condividendo la filosofia GNU GPL.

4.1.1 LA GNU GENERAL PUBLIC LICENSE

Come ogni licenza software, la GPL è un documento legale associato al programma rilasciato sotto tale licenza. Come ogni licenza di software libero, essa concede ai licenziatari il permesso di modificare il programma, di copiarlo e di ridistribuirlo con o senza modifiche, gratuitamente o a pagamento. Rispetto alle altre licenze di software libero, la GPL è classificabile come "persistente" e "propagativa".

È "persistente" perché impone un vincolo alla redistribuzione: se l'utente distribuisce copie del software, deve farlo secondo i termini della GPL stessa. In pratica, deve distribuire il testo della GPL assieme al software e corredarlo del codice sorgente o di istruzioni per poterlo ottenere ad un costo nominale. Questa è la caratteristica principe della GPL, il concetto ideato da Richard Stallman e da lui battezzato copyleft. L'effetto che realizza è l'esatto opposto di quello che produce invece il termine copyright: infatti il suo scopo è mantenere libero un programma una volta che esso è stato posto sotto GPL, anche se viene migliorato correggendolo e ampliandolo. Si noti che, escludendo la possibilità che l'output sia un lavoro derivato del programma, il copyleft si applica solo al software e non alla sua uscita.

È "propagativa" perché definisce nel testo una particolare interpretazione di "codice

derivato", tale che in generale l'unione di un programma coperto da GPL con un altro programma coperto da altra licenza può essere distribuita sotto GPL, o in alternativa non essere distribuita affatto. Nel primo caso si dice che l'altra licenza è "compatibile con la GPL"; nel secondo caso che non lo è. Questa caratteristica è indicata come strong copyleft nella terminologia della FSF. Il suo scopo è evitare che la persistenza venga via via indebolita apportando modifiche coperte da un'altra licenza meno libera, inficiando così lo scopo di mantenere libero il software coperto dalla GPL.

Ecco di seguito l'intestazione di tutti i file PHP del progetto:

```
* Copyright (C) 2008 Riccardo Cassinis

* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* For any further information please contact Riccardo Cassinis at
* riccardo.cassinis@unibs.it
* Department of Electronics for Automation
* Via Branze 38
* I-25123 Brescia (Italy)
* (+39) 030-371-5453
    *
```

4.1.2 NewEventoDB.php e TimeStamp.php

Queste sono i due sorgenti PHP più significativi del progetto. NewEventoDB è un'esempio di chiamata alla stored function NuovoEvento memorizzata all'interno del RDBMS che permette agli autori-attori del sistema di comunicare al DB Sauron l'occorrenza di un nuovo evento caratterizzato principalmente dall'istante di sistema nel quale è stato notificato che il sistema stesso riceve dalla chiamata a TimeStamp.PHP riportato di seguito. Tutti i sorgenti sono molto semplici ed esplicativi del funzionamento del progetto.

```
/*
* NewEventoDB.php
*
```

```
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`NuovoEvento`(2,2,'da php test
NewEvento',@data)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @data");
$id = mysql_result($result,0,0);
echo $id;
mysql_close($link);
?>
/*
* timestamp.php
*
* This script acts as a companion to photoreporter,
* adding date and time to the picture.
*
* R. Cassinis, August 2007
* Last modified January 16,2008 by M. Pescatori
*/
<?php
$src = "/tmp/pictures/webcam1.jpg";
$dst = "/tmp/pictures/webcam.jpg";
$tim = imagecreatefromjpeg($src);
//write current date and time
$headingcolor = imagecolorallocate($tim, 0,255,0);
$textcolor = imagecolorallocate($tim, 255, 255, 255);
// $textcolorbg = imagecolorallocate($tim, 0, 0, 0);
$timestring = date('D, d M Y H:i:s');
imagestring($tim, 3, 160, 0, php_uname('n') . " onboard camera", $headingcolor);
imagestring($tim, 3, 130, 225, "$timestring", $textcolor);
//save
imagejpeg($tim, $dst);
?>
```

4.1.3 NEWAUTOREDB.PHP E LEGGIIMMAGINEDB:PHP

Questi sono due esempi di chiamata delle stored procedures NuovoAutore e LeggiImmagine che rispettivamente inseriscono nel DB Sauron i dati di un nuovo autore di eventi e leggono una specifica immagine già inserita.

```
/*
* NewAutoreDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
```

```php
<?php
$db_host = '192.167.22.93';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`NuovoAutore`('mariopescatori',NULL,@data)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @data");
$id = mysql_result($result,0,0);
echo $id;
mysql_close($link);
?>
```

Sorgente SQL della procedura NuovoAutore invocata da NewAutore.php

```sql
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovoAutore`(IN nom TEXT, IN por INT,
OUT id INT)
BEGIN
INSERT INTO Autore SET Nome=nom , IndirizzoIP =USER() , Porta=por ;
SET id=LAST_INSERT_ID();
END$$
```

```php
/*
* LeggiImmagineDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$filename = 'letta.jpg';
$database = 'eventi';
$table = 'Immagine';
$id=36;
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`LeggiImmagine`( '$id', @dati , @comm)");
//Se leggo solo campo BLOB dati non funziona per problemi di terminazione!
if (!$result)
echo "CALL fallita" , mysql_error();
$result = mysql_query("SELECT @dati, @comm");
if (!$result)
echo "Manca parametro ritorno" , mysql_error();
$file = mysql_result($result,0,0);
echo mysql_result($result,0,1);
if (file_put_contents($filename, $file) === FALSE)
echo "Cannot write to file ($filename)";
mysql_close($link);
?>
```

Sorgente SQL della procedura LeggiImmagine invocata da LeggiImmagineDB.php

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `LeggiImmagine`(IN id INT, OUT datib
BLOB, OUT comm TEXT)
BEGIN
SELECT DatiBinari, Commento INTO datib, comm FROM Immagine WHERE ImmagineID=id;
END$$
```

4.1.4 PHP LancioRitornoFunzione e ImmaginaDB e Stored Function Invocate

```
/*
* LancioRitornoFunzione.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$filename = 'letta.jpg';
$database = 'eventi';
$table = 'Immagine';
$string = "`eventi`.`LaFunzione`('ciaoDum')";
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result=mysql_query("CALL".$string);
$result = mysql_query("SELECT".$string);
$id = mysql_result($result,0,0);
echo $id;
mysql_close($link);
?>
```

Sorgente SQL della funzione LaFunzione invocata da LancioRitornoFunzione.php

```
CREATE DEFINER=`root`@`localhost` FUNCTION `LaFunzione`(id TINYTEXT) RETURNS
int(11)
READS SQL DATA
BEGIN
INSERT INTO Autore SET Nome=id;
RETURN LAST_INSERT_ID();
END

/*
* ImmaginaDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$filename = 'letta.jpg';
$database = 'eventi';
$table = 'Immagine';
$string = "`eventi`.`Immagina`(3)";
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
```

```
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result=mysql_query("CALL".$string);
$result = mysql_query("SELECT".$string);
$id = mysql_result($result,0,0);
if (file_put_contents($filename, $id) === FALSE)
echo "Cannot write to file ($filename)";
mysql_close($link);
?>
```

Sorgente SQL della funzione Immagina invocata da ImmaginaDB.php

```
CREATE DEFINER=`root`@`localhost` FUNCTION `Immagina`(id INT) RETURNS blob
READS SQL DATA
BEGIN
DECLARE tmp BLOB;
SELECT DatiBinari INTO tmp FROM Immagine WHERE ImmagineID=id;
RETURN tmp;
END
```

4.1.5 UltimeImmaginiDB.php

```
/*
* UltimeImmaginiDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$id=43;
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("SELECT * FROM UltimeImmagini");
//Se leggo solo campo BLOB dati non funziona per problemi di terminazione!
if (!$result)
echo "CALL fallita" , mysql_error();
$i = 0;
while ($row = mysql_fetch_assoc($result))
{
$file = $row['DatiBinari'];
$filename = "/tmp/pictures/last/".$row['Tempo'].$row['Data'].".jpg";
if (file_put_contents($filename, $file) === FALSE)
echo "Cannot write to file ($filename)";
}
mysql_close($link);
?>
```

4.1.6 Altri Esempi di gestione dell'Interfaccia del DB con Sorgente PHP

```
/*
* NewTipoTabella.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`NuovoTipoEvento`('Immagine','Immagine
scattata da
telecamera',@data)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @data");
$id = mysql_result($result,0,0);
echo $id;
mysql_close($link);
?>


/*
* NewTipoEvento.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`NuovoTipoEvento`('Mappa','da
mapper3basic',@data)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @data");
$id = mysql_result($result,0,0);
echo $id;
mysql_close($link);
?>


/*
* NewMappa.php
*
* This script is a part of Sauron DB Project
```

```
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$filename = 'mappa2.map';
$autoreid = 12;
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$data = file_get_contents($filename);
$data = mysql_real_escape_string($data);
// Prepararazione dati usati in MySQL query
$result = mysql_query("CALL
`eventi`.`NuovaMappa`('$autoreid','$data','$filename','da php
NewMappa',@id)");
if (!$result)
echo "CALL fallita",mysql_error();
$result = mysql_query("SELECT @id");
if (!$result)
echo "Manca parametro ritorno" , mysql_error();
echo mysql_result($result,0,0);
mysql_close($link);
?>



*
* SalvaImmagineDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$data = file_get_contents('/tmp/pictures/webcam.jpg');
$data = mysql_real_escape_string($data);
// Prepararazione dati usati in MySQL query
$result = mysql_query("CALL `eventi`.`NuovaImmagine`(2,'$data','da php
NewImmagine',@id)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @id");
if (!$result)
echo "Manca parametro ritorno" , mysql_error();
echo mysql_result($result,0,0);
mysql_close($link);
?>
/*
* NewImmagine.php
```

```
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$data = file_get_contents('img.jpg');
$data = mysql_real_escape_string($data);
// Prepararazione dati usati in MySQL query
$result = mysql_query("CALL `eventi`.`NuovaImmagine`(2,'$data','da php
NewImmagine',@id)");
if (!$result)
echo "CALL fallita";
$result = mysql_query("SELECT @id");
if (!$result)
echo "Manca parametro ritorno" , mysql_error();
echo mysql_result($result,0,0);
mysql_close($link);
?>

/*
* LeggiMappaDB.php
*
* This script is a part of Sauron DB Project
*
* R. Cassinis, February 2008
* Last modified February 1,2008 by M. Pescatori
*/
<?php
$db_host = '192.0.2.5';
$db_user = 'sauron';
$db_pwd = 'sauron';
$database = 'eventi';
$table = 'Immagine';
$id=43;
$link = mysql_connect($db_host, $db_user, $db_pwd);
if (!$link)
die("Impossibile connettersi al database");
if (!mysql_select_db($database))
die("Impossibile selezionare tabella del database");
$result = mysql_query("CALL `eventi`.`LeggiMappa`( '$id', @datibinari ,
@nomefile)");
//Se leggo solo campo BLOB dati non funziona per problemi di terminazione!
if (!$result)
echo "CALL fallita" , mysql_error();
$result = mysql_query("SELECT @datibinari, @nomefile");
if (!$result)
echo "Manca parametro ritorno" , mysql_error();
$file = mysql_result($result,0,0);
$filename = mysql_result($result,0,1);
if (file_put_contents($filename, $file) === FALSE)
echo "Cannot write to file ($filename)";
mysql_close($link);
?>
```

4.2 Diagramma degli Oggetti del RDBMS Sauron

L'oggetto è l'istanza delle classi rappresentate dal modello del paragrafo 3.2.

Mentre la classe rappresenta un modello di una entità, l'oggetto è la sua determinata istanza, identificabile tramite i valori assunti dai suoi parametri o attributi. Data una classe, al tempo t possono "esistere" molteplici istanze, ossia oggetti, di quella classe. Di seguito vengono presentati i diagrammi degli oggetti del DB SAURON corredati dal corrispondente sorgente SQL di creazione.

4.2.1 Tabella Evento

È la tabella centrale del DataBase Sauron: memorizza ogni evento asincrono all'interno del sistema. Ogni istanza della tabella è univocamente identificata da una chiave primaria numerica che viene assegnata dal RDBMS all'atto della creazione invocata dalla stored function NuovoEvento che riceve dal client i dati da inserire nella tabella. Deve necessariamente validare la chiave esterna Autore e EventoID.

Evento

Commenti sulla tabella: Evento Asincrono trasmesso da Autore; InnoDB free: 5120 kB; (`Autore`) REFER `ev

Campo	Tipo	Null	Predefinito	
EventoID	bigint(20)	No		
Tempo	timestamp	No	CURRENT_TIMESTAMP	
Microsecondi	int(10)	No	0	
Autore	int(11)	No		
TipoEvento	int(11)	No		
Descrizione	text	Sì	*NULL*	

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	6861	Tempo
			Microsecondi
			Autore
			TipoEvento
EventoID	UNIQUE	6861	EventoID
DatiAutore	INDEX	4	Autore
DatiEvento	INDEX	2	TipoEvento

Spazio utilizzato:

Tipo	Utilizzo	
Dati	1.552	KiB
Indice	622.592	B
Totale	2.160	KiB

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	6.196
Lunghezza riga ø	256
Dimensione riga ø	357 B
Prossimo Autoindex	6.778
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `Evento` (
`EventoID` bigint(20) unsigned NOT NULL auto_increment,
`Tempo` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
`Microsecondi` int(10) unsigned NOT NULL default '0',
`Autore` int(11) NOT NULL,
`TipoEvento` int(11) NOT NULL,
`Descrizione` text,
PRIMARY KEY (`Tempo`,`Microsecondi`,`Autore`,`TipoEvento`),
UNIQUE KEY `EventoID` (`EventoID`),
KEY `DatiAutore` (`Autore`),
KEY `DatiEvento` (`TipoEvento`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Evento Asincrono trasmesso da Autore'
AUTO_INCREMENT=6779 ;
--
-- Limiti per la tabella `Evento`
--
ALTER TABLE `Evento`
ADD CONSTRAINT `DatiAutore` FOREIGN KEY (`Autore`) REFERENCES `Autore`
(`AutoreID`),
ADD CONSTRAINT `DatiEvento` FOREIGN KEY (`TipoEvento`) REFERENCES `TipoEvento`
(`TipoEventoID`);
--
```

--Illustrazione 12: Struttura e Limiti della Tabella Evento

Odometria

Commenti sulla tabella: Coordinate e orientamento robot; InnoDB free: 5120 kB; (`Mappa`) REFER `eventi/M

Campo	Tipo	Null	Predefinito	
OdometriaID	int(10)	No		
Ascissa	int(11)	No		
Ordinata	int(11)	No		
Angolo	float	No		
Mappa	int(11)	No		

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	0	OdometriaID
DatiMappa	INDEX	0	Mappa

Spazio utilizzato:

Tipo	Utilizzo	
Dati	16.384	B
Indice	16.384	B
Totale	32.768	B

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	0
Prossimo Autoindex	1
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `Odometria` (
`OdometriaID` int(10) unsigned NOT NULL auto_increment,
`Ascissa` int(11) NOT NULL,
`Ordinata` int(11) NOT NULL,
`Angolo` float NOT NULL,
`Mappa` int(11) NOT NULL,
PRIMARY KEY (`OdometriaID`),
KEY `DatiMappa` (`Mappa`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Coordinate e orientamento
robot' AUTO_INCREMENT=1 ;
_
-- Limiti per la tabella `Odometria`
--
ALTER TABLE `Odometria`
ADD CONSTRAINT `DatiMappa` FOREIGN KEY (`Mappa`) REFERENCES `Mappa`
(`MappaID`);
--
```

Illustrazione 13: Struttura e Limiti della Tabella Odometria

Immagine

Commenti sulla tabella: Memorizza file binario dell'Evento Immagine; InnoDB free: 5120 kB

Campo	Tipo	Null	Predefinito	
ImmagineID	int(11)	No		
DatiBinari	blob	No		
Commento	text	No		

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	41439	ImmagineID

Spazio utilizzato:

Tipo	Utilizzo	
Dati	126.480	KiB
Indice	0	B
Totale	126.480	KiB

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	41.439
Lunghezza riga ø	3.125
Dimensione riga ø	3.125 B
Prossimo Autoindex	6.778
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `Immagine` (
`ImmagineID` int(11) NOT NULL auto_increment,

`DatiBinari` blob NOT NULL,
`Commento` text NOT NULL,
PRIMARY KEY (`ImmagineID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Memorizza file binario dell''Evento
Immagine'
AUTO_INCREMENT=6779 ;
```

Illustrazione 14: Struttura Tabella Immagine

Mappa

Commenti sulla tabella: Descrizione Ambiente Lavoro; InnoDB free: 5120 kB

Campo	Tipo	Null	Predefinito	
MappaID	int(11)	No		
NomeFile	tinytext	No		
Creazione	timestamp	No	CURRENT_TIMESTAMP	
Modifica	timestamp	No	0000-00-00 00:00:00	
Descrizione	text	No		
DatiBinari	blob	Sì	*NULL*	

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	10	MappaID

Spazio utilizzato:

Tipo	Utilizzo	
Dati	65.536	B
Indice	0	B
Totale	65.536	B

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	4
Lunghezza riga ø	16.384
Dimensione riga ø	16.384 B
Prossimo Autoindex	45
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `Mappa` (
`MappaID` int(11) NOT NULL auto_increment,
`NomeFile` tinytext NOT NULL,
`Creazione` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
`Modifica` timestamp NOT NULL default '0000-00-00 00:00:00',
`Descrizione` text NOT NULL,
`DatiBinari` blob,
PRIMARY KEY (`MappaID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Descrizione Ambiente Lavoro'
AUTO_INCREMENT=45 ;
```

Illustrazione 15: Struttura Tabella Mappa

Robot

Commenti sulla tabella: Descrive unità robotiche inserite nel sistema Sauron; InnoDB free: 5120 kB

Campo	Tipo	Null	Predefinito	
RobotID	int(11)	No		
Nome	text	No		
Descrizione	text	Sì	*NULL*	
Porta	int(11)	Sì	*NULL*	
IndirizzoIP	text	Sì	*NULL*	

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	0	RobotID

Spazio utilizzato:

Tipo	Utilizzo	
Dati	16.384	B
Indice	0	B
Totale	16.384	B

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	0
Prossimo Autoindex	1
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `Robot` (
`RobotID` int(11) NOT NULL auto_increment,
`Nome` text NOT NULL,
`Descrizione` text,
`Porta` int(11) default NULL,
`IndirizzoIP` text,
PRIMARY KEY (`RobotID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Descrive unità robotiche inserite
nel sistema Sauron'
AUTO_INCREMENT=1 ;
```

Illustrazione 16: Struttura Tabella Robot

VediEventi

Commenti sulla tabella: VIEW

Campo	Tipo	Null	Predefinito	
EventoID	bigint(20)	No	0	
Tempo	timestamp	No	0000-00-00 00:00:00	
IndirizzoIP	mediumtext	Sì	*NULL*	
NomeTabella	tinytext	No		

```
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER
VIEW `eventi`.`VediEventi`
AS select `eventi`.`Evento`.`EventoID` AS `EventoID`,`eventi`.`Evento`.`Tempo`
AS
`Tempo`,`eventi`.`Autore`.`IndirizzoIP` AS
`IndirizzoIP`,`eventi`.`TipoEvento`.`NomeTabella` AS
`NomeTabella` from ((`eventi`.`Evento` join `eventi`.`Autore`) join
`eventi`.`TipoEvento`) where
((`eventi`.`Evento`.`Autore` = `eventi`.`Autore`.`AutoreID`) and
(`eventi`.`Evento`.`TipoEvento` =
`eventi`.`TipoEvento`.`TipoEventoID`));
```

Illustrazione 17: Struttura Vista VediEventi

TipoErrore

Commenti sulla tabella: Codice e Descrizione Errori di Sistema; InnoDB free: 5120 kB

Campo	Tipo	Null	Predefinito	
ErroreID	tinyint(4)	No		
Descrizione	tinytext	No		
DataCrezione	timestamp	No	CURRENT_TIMESTAMP	
Commenti	tinytext	No		

Indici:

Nome chiave	Tipo	Cardinalità	Campo
PRIMARY	PRIMARY	0	ErroreID

Spazio utilizzato:

Tipo	Utilizzo	
Dati	16.384	B
Indice	0	B
Totale	16.384	B

Statistiche righe:

Istruzioni	Valore
Formato	Compact
Righe	0
Prossimo Autoindex	1
Creazione	01 Feb, 2008 at 01:05 PM

```
CREATE TABLE `TipoErrore` (
`ErroreID` tinyint(4) NOT NULL auto_increment,
`Descrizione` tinytext NOT NULL,
`DataCrezione` timestamp NOT NULL default CURRENT_TIMESTAMP on update
CURRENT_TIMESTAMP,
`Commenti` tinytext NOT NULL,
PRIMARY KEY (`ErroreID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Codice e Descrizione Errori
di Sistema'
AUTO_INCREMENT=1 ;
```

Illustrazione 18: Struttura Tabella TipoErrore

UltimeImmagini

Commenti sulla tabella: VIEW

Campo	Tipo	Null	Predefinito	
DatiBinari	blob	No		
Tempo	time	Sì	*NULL*	
Data	date	Sì	*NULL*	

```
--
-- Struttura della tabella `UltimeImmagini` è una tabella tipo VIEW
--
CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`%` SQL SECURITY DEFINER VIEW
`eventi`.`UltimeImmagini` AS
select `eventi`.`Immagine`.`DatiBinari` AS
`DatiBinari`,cast(`eventi`.`Evento`.`Tempo` as time) AS
`Tempo`,cast(`eventi`.`Evento`.`Tempo` as date) AS `Data` from (`eventi`.`Immagine`
join
`eventi`.`Evento`) where (`eventi`.`Evento`.`EventoID` =
`eventi`.`Immagine`.`ImmagineID`) order by
`eventi`.`Evento`.`Tempo` desc limit 20;
```

Illustrazione 19: Struttura Vista UltimeImmagini

4.3 Procedure e Funzioni di Accesso ai Dati Memorizzate e Gestite Direttamente dal RDBMS

In questo codice MySQL sono definite procedure e funzioni memorizzate nel database Sauron che vengono chiamate dai sorgenti PHP dei paragrafi precedenti o possono essere inserite in uno qualsiasi dei linguaggi di programmazione per cui MySQL prevede le librerie di interfacciamento. Sono il cuore del sistema, permettono di proteggere i dati contenuti nel RDBMS da accessi maldestri e ne velocizzano il funzionamento. Al tempo stesso facilitano l'aggiornamento e la manutenzione del Sw evitando duplicazioni ed aggiornamenti di tutta la rete dell'ARL.

```
-- Procedure
--
-- DELIMITER $$
--
CREATE DEFINER=`root`@`localhost` PROCEDURE `EliminaAutore`(IN id INT)
BEGIN
DELETE FROM Autore WHERE AutoreID=id;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `EliminaEvento`(IN id INT)
BEGIN
DELETE FROM Evento WHERE EventoID=id;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `EliminaImmagine`(IN id INT)
BEGIN
START TRANSACTION;
CALL `eventi`.`EliminaEvento`(id);
DELETE FROM Immagine WHERE ImmagineID=id;
COMMIT;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `EliminaMappa`(IN id INT)
BEGIN
START TRANSACTION;
CALL `eventi`.`EliminaEvento`(id);
DELETE FROM Mappa WHERE MappaID=id;
COMMIT;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `EliminaTipoEvento`(IN id INT)
BEGIN
DELETE FROM TipoEvento WHERE TipoEventoID=id;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `LeggiImmagine`(IN id INT, OUT datib
BLOB, OUT comm TEXT)
BEGIN
SELECT DatiBinari, Commento INTO datib, comm FROM Immagine WHERE ImmagineID=id;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `LeggiMappa`(IN id INT, OUT datib
BLOB, OUT name TEXT)
BEGIN
SELECT DatiBinari, NomeFile INTO datib, name FROM Mappa WHERE MappaID=id;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovaImmagine`(IN autoreid INTEGER,
IN datib BLOB, IN
comm TEXT,OUT id INTEGER)
BEGIN
```

```
START TRANSACTION;
CALL `eventi`.`NuovoEvento`(autoreid,1,'da stored procedure NuovaImmagine',id);
INSERT INTO Immagine SET ImmagineID=id, DatiBinari=datib,Commento=comm;
COMMIT;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovaMappa`(IN autoreid INTEGER, IN
datib BLOB, IN name
TEXT, IN comm TEXT,OUT id INTEGER)
BEGIN
START TRANSACTION;
CALL `eventi`.`NuovoEvento`(autoreid,2,'da stored procedure NuovaMappa',id);
INSERT INTO Mappa SET MappaID=id, DatiBinari=datib, NomeFile=name,
Descrizione=comm;
Commit;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovoAutore`(IN nom TEXT, IN por INT,
OUT id INT)
BEGIN
INSERT INTO Autore SET Nome=nom , IndirizzoIP =USER() , Porta=por ;
SET id=LAST_INSERT_ID();
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovoEvento`(IN autoreid INT, IN
tipoeventoid INT, IN
commento TEXT,OUT id INT)
BEGIN
START TRANSACTION;
INSERT INTO `Evento` ( `EventoID` , `Tempo` , `Microsecondi` , `Autore` ,
`TipoEvento` ,
`Descrizione` ) VALUES (NULL , NOW( ) , MICROSECOND(NOW_USEC()),autoreid,
tipoeventoid, commento);
SET id=LAST_INSERT_ID();
COMMIT;
END$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `NuovoTipoEvento`(IN nome TINYTEXT, IN
descr TEXT, OUT id
INT)
BEGIN
START TRANSACTION;
INSERT INTO TipoEvento SET NomeTabella=nome , Descrizione=descr ;
SET id=LAST_INSERT_ID();
COMMIT;
END$$
--
-- DELIMITER ;
--
-- Funzioni
--
CREATE DEFINER=`root`@`localhost` FUNCTION `Immagina`(id INT) RETURNS blob
READS SQL DATA
BEGIN
DECLARE tmp BLOB;
SELECT DatiBinari INTO tmp FROM Immagine WHERE ImmagineID=id;
RETURN tmp;
END
CREATE DEFINER=`root`@`localhost` FUNCTION `LaFunzione`(id TINYTEXT) RETURNS
int(11)
READS SQL DATA
BEGIN
INSERT INTO Autore SET Nome=id;
RETURN LAST_INSERT_ID();
END
```

CAPITOLO 5 – RISULTATI SPERIMENTALI

I brevi test eseguiti per verificare in fase di progettazione il funzionamento del RDBMS Sauron all'interno del sistema si sono limitati a verificare la capacità di memorizzare il dato più voluminoso costituito nel caso di studio dalle immagini spedite dalla webcam. Il DB Sauron riusciva a gestire il flusso dati senza problemi e quindi a soddisfare le specifiche di progettazione. Questo mostrando potenzialità ben superiori tutte da verificare. La fase di test del RDBMS Sauron non è mai stata eseguita.

CAPITOLO 6 – CONCLUSIONI ED EVOLUZIONE

In futuro se venisse reinstallato il DB Sauron avrebbe interessanti prospettive di evoluzione rispetto a:

- sicurezza degli accessi con una opportuna progettazione delle politiche di gestione della risorsa DB, tempi e spazi da mettere a disposizione dei vari utenti
- suddivisione delle risorse per aumentare la velocità si servizio a utenti real-time
- sicurezza dei dati con gestione di copie di backup o distribuzione del DB su rete di server per scongiurare crash
- assegnazione ai vari utenti collegati di un canale di comunicazione caratterizzato da un volume massimo di dati da trasmettere al DB (bit-rate)
- nuove funzionalità per le tabelle di accesso e consultazione on-line con strumento di visualizzazione via web

Il progetto si è rivelato un utile approfondimento degli studi della mia carriera universitaria applicati ad un sistema di indubbio interesse sia scientifico che sperimentale.

APPENDICE A – ANALISI DETTAGLIATA DEL FUNZIONAMENTO DELLE STORED PROCEDURES

Le **stored procedures** (procedure memorizzate) sono un'altra delle caratteristiche la cui assenza è stata a lungo sottolineata dai detrattori di MySQL: con la versione 5.0 si è finalmente posto rimedio a questa assenza.

Una stored procedure è un insieme di istruzioni SQL che vengono memorizzate nel server con un nome che le identifica; tale nome consente in seguito di rieseguire l'insieme di istruzioni facendo semplicemente riferimento ad esso. Vediamo come creare una stored procedure:

```
    CREATE PROCEDURE nome ([parametro[,...]])
[SQL SECURITY { DEFINER | INVOKER }] corpo
//parametri:
[ IN | OUT | INOUT ] nomeParametro tipo
```

Come abbiamo detto, ogni procedura è identificata da un nome. Inoltre la procedura è attribuita ad uno specifico database (a partire da MySQL 5.0.1), esattamente come una tabella. Di conseguenza la procedura viene assegnata ad un database al momento della creazione, ed i nomi referenziati al suo interno si riferiranno allo stesso database, a meno che non siano qualificati con un nome di database specifico. In fase di creazione, quindi, se indichiamo il nome della procedura senza specificare il database questa sarà assegnata al db attualmente in uso.

Ogni procedura può avere uno o più parametri, ciascuno dei quali è formato da un nome, un tipo di dato e l'indicazione se trattasi di parametro di input, di output o entrambi. Se manca l'indicazione, il parametro è considerato di input.

La clausola SQL SECURITY stabilisce se, al momento dell'esecuzione, la procedura utilizzerà i permessi dell'utente che la sta eseguendo o quelli dell'utente che l'ha creata (il default è DEFINER).

Vediamo adesso un esempio concreto di stored procedure:

```
CREATE PROCEDURE nomeProc (IN param1 INT, OUT param2 INT)
```

```
SELECT COUNT(*) INTO param2 FROM tabella
WHERE campo1 = param1;
```

Questa istruzione crea una procedura chiamata 'nomeProc' nel database in uso; la procedura usa un parametro in input e uno in output, entrambi interi, ed effettua il conteggio delle righe in tabella in cui il valore di campo1 corrisponde al primo parametro; il risultato della query viene memorizzato nel secondo parametro attraverso la clausola INTO.

Ecco come chiamare la procedura e visualizzare il risultato:

```
    CALL nomeProc (5, @a);
SELECT @a;
```

Con l'istruzione CALL effettuiamo la chiamata della procedura (immaginando che il database attualmente in uso sia lo stesso a cui la procedura è associata), passando il valore 5 come parametro di input e la variabile @a come parametro di output, nel quale verrà memorizzato il risultato. La SELECT successiva visualizza il valore di tale variabile dopo l'esecuzione.

Nell'esempio appena visto la stored procedure conteneva una semplice SELECT; è possibile invece creare procedure che contengono sintassi complesse comprendenti più istruzioni: in pratica, dei veri e propri script, con la possibilità di controllare il flusso attraverso vari costrutti (IF, CASE, LOOP, WHILE, REPEAT, LEAVE, ITERATE). Inoltre è possibile utilizzare i **cursori** per gestire i resultset.

Ecco un esempio di procedura di una certa complessità:

```
DELIMITER //
CREATE PROCEDURE procedura1 (param1 INT, param2 CHAR(3),
        OUT param3 INT)
BEGIN
        DECLARE finito INT default 0;
        DECLARE a INT;
        DECLARE b CHAR(50);
        DECLARE cur1 CURSOR FOR SELECT id,nome
                FROM clienti WHERE cat = param2;
        DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
                SET finito = 1;
        OPEN cur1;
        SET param3 = 0;
        FETCH cur1 INTO a,b;
        ciclo: WHILE NOT finito DO
                IF param3 < param1 THEN
                        SET param3 = param3 + 1;
```

```
                        FETCH cur1 INTO a,b;
                ELSE
                        LEAVE ciclo;
                END IF;
        END WHILE ciclo;
END; //
DELIMITER ;
```

Analizziamo questo codice.

Per prima cosa notiamo il comando **DELIMITER**, che serve al client mysql per modificare il normale delimitatore delle istruzioni, che sarebbe il punto e virgola. Infatti, siccome la stored procedure contiene più istruzioni, al suo interno il punto e virgola viene utilizzato più volte. Di conseguenza, se vogliamo riuscire a memorizzare la procedura, dobbiamo comunicare al server che il delimitatore è un altro; in caso contrario, al primo punto e virgola penserebbe che la nostra CREATE sia terminata. Decidiamo quindi che il nuovo delimitatore sia un doppio slash.

Dichiariamo la procedura col nome *procedura1* e tre parametri, di cui i primi due di input e l'ultimo di output. Il codice da eseguire deve essere racchiuso tra le clausole **BEGIN** ed **END**; all'interno troviamo tre istruzioni **DECLARE** che dichiarano altrettante **variabili**, quindi la dichiarazione di un **cursore**, infine la dichiarazione di un **HANDLER** per l'SQLSTATE 02000, di cui vedremo tra poco l'utilità.

La prima vera operazione è l'apertura del cursore (**OPEN**), che esegue la query associata utilizzando uno dei parametri di input. Di seguito, dopo avere inizializzato il parametro di output (che servirà da contatore delle righe lette), eseguiamo la prima **FETCH**, cioè la lettura della prima riga di risultato della query, i cui valori sono assegnati alle variabili a e b. Di seguito c'è un ciclo **WHILE** (al quale abbiamo assegnato proprio il nome 'ciclo') che viene eseguito fintanto che il valore di 'finito' è falso, cioè è uguale a zero come da inizializzazione. Tale valore verrà modificato nel momento in cui non ci sono più righe da leggere, come specificato con l'handler (02000 è infatti lo stato che indica tale situazione).

All'interno del ciclo si verifica se la variabile 'param3' ha già raggiunto il valore di

'param1', che era l'altro parametro in input: in questo caso l'istruzione **LEAVE** consente di abbandonare il ciclo stesso; in caso contrario si incrementa la variabile e si esegue una nuova lettura del cursore, sempre riportando i risultati nelle variabili a e b (l'esempio è solo dimostrativo, in quanto non utilizziamo mai questi valori).

Il ciclo quindi termina quando sono finite le righe del cursore oppure quando ne abbiamo lette tante quante indicate dal primo parametro, a seconda di quale dei due eventi si verifica prima. Al termine dell'esecuzione il parametro di output conterrà il numero di righe lette. L'istruzione DELIMITER viene utilizzata di nuovo per ripristinare il delimitatore punto e virgola, una volta che la procedura è stata memorizzata.

Vediamo quindi quali elementi possiamo trovare in una stored procedure:

- **Variabili**: sono quelle dichiarate con l'istruzione DECLARE, più i parametri ricevuti dalla procedura
- **Condizioni**: corrispondono a codici errore di MySQL oppure valori di SQLSTATE ai quali possiamo dare un nome per poi gestirli con un HANDLER
- **Cursori**: si usano per eseguire query che restituiscono un resultset quando vogliamo scorrere tale resultset all'interno della procedura
- **Handler**: istruzioni da eseguire al verificarsi di particolari condizioni; tali condizioni possono essere, oltre a quelle definite in precedenza, direttamente un SQLSTATE, oppure un codice errore o altre condizioni: SQLWARNING, NOT FOUND, SQLEXCEPTION. La dichiarazione deve anche specificare se, al verificarsi della condizione, la procedura deve continuare o terminare (CONTINUE o EXIT)

Tutti questi elementi sono creati con l'istruzione DECLARE, e devono trovarsi all'inizio del codice (dopo BEGIN) nell'ordine in cui li abbiamo elencati.

Vediamo ora cosa possiamo trovare nel codice vero e proprio:

- **Variabili**: possono essere manipolate con l'istruzione **SET** o ricevere i valori delle query con **INTO**

- **Cursori**: si aprono con **OPEN**, si chiudono con **CLOSE** (MySQL li chiude in automatico al termine della procedura se non lo facciamo esplicitamente), e si scorrono con **FETCH**, che provvede a leggere una riga e mettere i risultati in una o più variabili
- **Controllo di flusso**: sono costrutti che ci permettono di inserire una logica nella procedura. Abbiamo **IF** per le condizioni, **LOOP**, **REPEAT** e **WHILE** per i cicli, **ITERATE** e **LEAVE** per la gestione dall'interno dei cicli stessi

Vediamo un po' di codice di esempio:

```
DECLARE a INT default 0;
DECLARE cond1 CONDITION FOR 1045;
DECLARE cond2 CONDITION FOR SQLSTATE '02000';
DECLARE cur1 CURSOR FOR query;
DECLARE EXIT HANDLER FOR cond1 SET variabile = valore;
DECLARE CONTINUE HANDLER FOR cond2 SET variabile = valore;
DECLARE CONTINUE HANDLER FOR SQLWARNING;
OPEN cur1;
FETCH cur1 INTO variabile1 variabile2;
CLOSE cur1;
IF condizione THEN istruzioni
    [ELSEIF condizione THEN istruzioni] ...
    [ELSE istruzioni]
END IF;
[nome:] LOOP
    istruzioni
END LOOP [nome];
[nome:] REPEAT
    istruzioni
    UNTIL condizione
END REPEAT [nome];
[nome] WHILE condizione DO
    istruzioni
END WHILE [nome];
ITERATE nomeCiclo;
LEAVE nomeCiclo;
```

Nella prima riga definiamo una variabile; nella seconda diamo un nome al codice errore 1045; nella terza all'SQLSTATE '02000'; nella quarta definiamo un cursore. Di seguito definiamo gli handler per le due condizioni definite in precedenza, specificando che nel caso della prima interromperemo la procedura, mentre nel secondo la continueremo; in entrambi i casi impostiamo il valore di una variabile. Definiamo poi un handler anche per la condizione di SQLWARNING. Le tre istruzioni successive effettuano apertura, lettura e chiusura del cursore.

Abbiamo poi la sintassi della IF, seguita da quelle relative ai cicli; possiamo notare che il ciclo LOOP non ha condizione di uscita (dovremo quindi uscirne con un LEAVE o attraverso un handler), mentre il ciclo REPEAT ha una condizione di uscita (cioè termina quando la condizione è vera), ma viene comunque eseguito almeno una volta; il ciclo WHILE ha una condizione di permanenza (termina quando diventa falsa), e può anche non essere eseguito mai.

Con LEAVE abbandoniamo un ciclo, mentre con ITERATE passiamo all'esecuzione successiva (saltiamo cioè, per una esecuzione, la parte di codice interna al ciclo successiva alla ITERATE). I nomi dei cicli sono facoltativi, ma diventano indispensabili qualora dobbiamo referenziarli con LEAVE o ITERATE.

Una volta creata, la stored procedure può essere eliminata con **DROP PROCEDURE**, o modificata con **ALTER PROCEDURE**. Non è tuttavia possibile modificare il codice della procedura: per fare questo dovremo eliminarla e ricrearla.

```
ALTER PROCEDURE nomeProcedura SQL SECURITY { DEFINER | INVOKER };
DROP PROCEDURE nomeProcedura;
```

A.1 I PERMESSI NECESSARI

Per creare una stored procedure dobbiamo avere il privilegio CREATE ROUTINE, nonché il privilegio SUPER in caso siano abilitati i log binari di MySQL. Per modificarla o eliminarla ci serve il privilegio ALTER ROUTINE (che viene assegnato automaticamente al creatore della procedura). Per eseguirla infine è necessario il privilegio EXECUTE (anche questo assegnato in automatico all'autore) nonché il permesso di accesso al database che la contiene.

I controlli sulle istruzioni contenute dipendono dal valore impostato per il parametro SQL SECURITY: nel caso sia DEFINER (il default), sarà l'utente che ha definito la procedura che deve avere i permessi necessari; se invece il valore del parametro è INVOKER i permessi sono richiesti a chi esegue la routine.

A.2 LE STORED FUNCTIONS

Le stored functions sono simili alle stored procedures, ma hanno uno scopo più semplice, cioè quello di definire vere e proprie funzioni, come quelle già fornite da MySQL. Esse restituiscono un valore, e non possono quindi restituire resultset, al contrario delle stored procedures. Nelle versioni di MySQL precedenti alla 5.0 esistevano le "user-defined functions", che venivano memorizzate esternamente al server. Ora queste funzioni sono ancora supportate, ma è sicuramente consigliabile utilizzare le nuove stored functions.

Vediamo come crearle:

```
CREATE FUNCTION nome ([parametro[,...]])
RETURNS tipo
[SQL SECURITY { DEFINER | INVOKER }] corpo
//parametri:
nomeParametro tipo
```

Rispetto alle stored procedures, vediamo che si aggiunge la clausola RETURNS che specifica che tipo di dato la funzione restituisce. Inoltre nella lista dei parametri non esiste distinzione fra input e output, in quanto i parametri sono solo in input.

Esistono poi le istruzioni **ALTER FUNCTION** e **CREATE FUNCTION** che sono analoghe alle corrispondenti relative alle procedure. Anche le considerazioni sulla sicurezza sono le stesse relative alle stored procedures. Il codice contenuto può essere lo stesso, con la differenza che, come abbiamo già detto, non sono previsti parametri in output e non è possibile restituire un resultset. é invece **obbligatorio** restituire un dato del tipo indicato nella clausola RETURNS, attraverso l'istruzione **RETURN** *valore*.

Per informazioni più approfondite su stored procedures e stored functions vi rimandiamo al manuale ufficiale di MySQL.

A.3 Gestire le Stored Procedure e Stored Functions con MySQL Query Browser

Il MySQL Query Browser supporta Stored Procedures e Stored Functions se usato con MySQL 5 o superiore. E' possibile creare, modificare e visualizzare stored procedure e stored function.

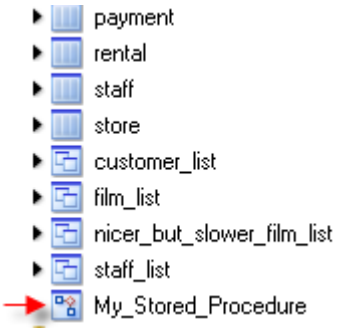


Illustrazione 20: Stored procedure nel browser DB

Le stored procedure e le stored function sono contrassegnate, nel browser dei database, con una icona speciale per distinguerle dalle normali tabelle e viste. Cliccate sulla freccia situata alla sinistra di una stored procedure o di una stored function per visualizzare la lista di parametri per la suddetta procedura o funzione.

Per modificare una stored procedure o una stored function, fate clic destro su di essa nel browser dei database e scegliete l'opzione Edit Procedure oppure Edit Function. Questo aprirà una nuova linguetta dell'editor degli script, contenente la procedura o la funzione scelte.

Dopo che avrete finito di modificarla, cliccate sul pulsante Execute al di sopra dell'area degli script, oppure cliccate sull'opzione Execute del menu Script per aggiornare la procedura o la funzione.

Per creare una nuova stored procedure o function, scegliete l'opzione Create Stored Procedure / Function dal menu Script. Inserite il nome della procedura o della funzione e cliccate sul pulsante Create PROCEDURE o Create FUNCTION. Verrà creato un modello simile al seguente:

```
DELIMITER \\

DROP PROCEDURE IF EXISTS `test`.`new_proc`\\
CREATE PROCEDURE `test`.`new_proc` ()
BEGIN

END\\
```

```
DELIMITER ;
```

Dopo aver inserito il codice della procedura/funzione, per crearla cliccate sul pulsante Execute al di sopra dell'area degli script, oppure scegliete la voce Execute del menu Script.

Per modificare tutte le stored procedure/function in una sola volta, scegliete l'opzione Edit All Stored Procedures / Functions dal menu Script. Comparirà una nuova linguetta per la modifica degli script, contenente tutte le stored procedure e le stored function per il database di default corrente.

Per cancellare una stored procedure o una stored function esistente, fate clic destro su di essa nel browser dei database e scegliete l'opzione Drop Procedure oppure Drop Function.

APPENDICE B – SORGENTE PHOTOREPORTER-V2.1

Scatta una serie di immagini, una ogni secondo e le memorizza nel database Sauron sul calcolatore denominato Rosie ed in copia in /tmp/foto/last_image_xx.jpg, dove xx varia da 00 al numero specificato-1 (49 di default). Esegue all'infinito il loop e quando l'array di file è carico ricomincia ricaricando l'immagine /tmp/pictures/last_image_00.jpg.

Le immagini sono prese dal dispositivo di input specificato, e conservate nel formato sif (320 x 240)

Richiede una directory /tmp/immagini/ con i permessi rwxrwxrwx, che dovrebbe inizialmente contenere un file chiamato black_image_sif.jpg, un fondo nero con la scritta immagine non disponibile. Usa anche timestamp.php, presentato al paragrafo 4.1.2, ed i programmi vgrabbj e setpwc.

```
#!/bin/bash
# Copyright (C) 2007 Riccardo Cassinis
#
# R. Cassinis, September, 2007
# Last modified February 1,2008 by M. Pescatori
# Standard parameters:
inputdevice="/dev/video0"
#numimages=10
#timelapse=1
reverse=""
lockfile="/tmp/pictures/photoreporter"
usage1="Usage: photoreporter [-h] [-d device] [-n numinmages] [-t timelapse] [-r]"
usage2=" photoreporter -k"
#php apridb.php
function diavoletto
# This is the daemon that keeps running when the script is done.
# Kill it with photoreporter -k
{
# Keep grabbing images
while [ 1 ]
do
vgrabbj -i sif -f /tmp/pictures/webcam.jpg -d $inputdevice $reverse -n
# Add timestamp and caption
# php /tmp/pictures/timestamp.php
php SalvaImmagineDB.php
# Copy each image to archive
# cp /tmp/pictures/webcam.jpg /tmp/pictures/$NOMEFILE
# Rest for a while
# sleep $timelapse #valore del parametro dello script
# done
done
}

while getopts "hd:t:n:rk" options; do
case $options in
d ) inputdevice=$OPTARG
echo "device is $inputdevice";;
h ) echo $usage1
```

```
echo $usage2
exit 1;;
n ) numimages=$OPTARG
echo "Acquiring $numimages images";;
t ) timelapse=$OPTARG
echo "Acquiring images every $timelapse seconds";;
r ) reverse="-U -R"
echo "Acquiring reversed images";;
\? ) echo $usage1
echo $usage2
exit 1;;
k ) cp /tmp/pictures/black_image_sif.jpg /tmp/pictures/webcam.jpg
rm -rf $lockfile
# This is a suicide: must be the last operation performed
killall -q photoreporter
exit 0;;
esac
done
# First, let's make sure that no other instances of this script are running:
if [ -f $lockfile ]
then
echo [ERROR] Lock file found! Is another instance of this script running?
exit 1
fi
# Second, let's make sure that the video device exists and is available:
if [ ! -c $inputdevice ]
then
echo [ERROR] Video device does not exist - exiting
exit 1
fi
string1=$(fuser $inputdevice 2>/dev/null)
if [ $string1 ]
then
echo [ERROR] Video device $inputdevice is busy - exiting
exit 1
fi
# Set auto exposure and auto shutter speed
setpwc -d $inputdevice -g -1 -s -1
# Grab a few images and discard them, to allow auto exposure to settle
vgrabbj -i sif -z 20 -f /tmp/pictures/xxx.jpg -d $inputdevice -U -R
rm -f /tmp/pictures/xxx.jpg
# Let's create a lock file
touch $lockfile
# Let's start daemon
diavoletto &
exit 0
```

BIBLIOGRAFIA

- Cassinis, R., Meriggi, P., Panteghini, M.: "A Very Low Cost Distributed Localization and Navigation System for a Mobile Robot", Proc. ECMR '03, Radziejowice, Poland, 2003
- Cassinis, R., Tampalini, F., Bartolini, P.: "Wireless Network Issues for a Roaming Robot", Proc. ECMR '05, Ancona, Italy, 2005
- Cassinis, R., Tampalini, F., Fedrigotti, R.: "Active Markers for Outdoor and Indoor Robot Localization" Proc. TAROS 2005, London, UK, 2005
- Cassinis, R., Tampalini, F., Bartolini, P., Fedrigotti, R.: "Docking and Charging System for Autono-mous Mobile Robots", Technical Report, DEA-Unibs, 2005
- Cassinis, R., Tampalini, F.: "Fuzzy Logic Controller based on XML-formatted Files for Behaviour-based Mobile Robots", Technical Report, DEA-Unibs, 2005
- Cassinis, R., Tampalini, F.: "ARCSUS - Advanced Robotic Control System Using SVG", ICGST-ARAS Journal, Vol. 5, N. 2, Pag. 19-25, Cairo, Egypt, 2006