

# Introduzione a MySQL

Prof. Alfredo Pulvirenti  
Prof. Salvatore Alaimo

# Sommario

- Introduzione a MySQL
  - Engine
  - Tipi di Dati
  - Installazione di MySQL
- Programmi Client
- Account e Privilegi
- Comandi MySQL
  - Alcuni esempi pratici
- Funzioni e Operatori
  - Altri esempi pratici
- Stored Procedure
- Special Topics

# Un po' di storia

- Creato dalla società MySQLAB sin dal 1979 soltanto dal 1996 supporta anche SQL.
- Sun Microsystem nel 2008 rileva la società per 1 miliardo di dollari



- Nel 2010 Oracle acquista Sun per 7,5 miliardi di dollari possedendo così anche MySQL.

**ORACLE®**

# Cosa è MySQL

- MySQL, definito Oracle MySQL, è un Relational Database Management System, composto da un client con interfaccia a riga di comando e un server, entrambi disponibili sia per sistemi Unix o Unix-like come GNU/Linux che per Windows, anche se prevale un suo utilizzo in ambito Unix.
- Oggi l'ultima versione disponibile è la 8.0.x (MariaDB 10.9 a breve 10.10)
- Dal 1996 supporta la maggior parte della sintassi SQL e si prevede in futuro il pieno rispetto dello standard ANSI.
- Possiede delle interfacce per diversi linguaggi, compreso un driver ODBC, due driver Java, un driver per Mono e .NET ed una libreria per Python.

# Engine

- MySQL mette a disposizione diversi **tipi di tabelle** (“**storage engine**”) per la memorizzazione dei dati.
- Ognuno presenta proprietà e caratteristiche differenti.
- Esiste una **API** che si può utilizzare per creare nuovi tipi di tabella che si possono installare senza necessità di riavviare il server.
- Due sono i sistemi principali:
  - **Transazionali**: sono più sicuri, permettono di recuperare i dati anche in caso di crash, e consentono di effettuare modifiche tutte insieme;
  - **Non transazionali**: sono più veloci, occupano meno spazio su disco e minor richiesta di memoria.

# Engine: Mylsam

- **MyISAM** era lo storage engine di default dal MySQL 3.23 fino al MySQL 5.4.
- **MyISAM** utilizza la struttura **ISAM** e deriva da un tipo più vecchio, oggi non più utilizzato, che si chiamava appunto **ISAM**.
- È un motore di immagazzinamento dei dati estremamente veloce e richiede poche risorse, sia in termini di memoria RAM, sia in termini di spazio su disco.
- Il suo limite principale rispetto ad alcuni altri SE consiste nel mancato supporto delle transazioni e alle foreign key.
- Ogni tabella MyISAM è memorizzata all'interno del disco con tre file:
  - un file **.frm** che contiene la definizione della tabella,
  - un file **.MYD** per i dati
  - un file **.MYI** per gli indici

## Engine: InnoDB

- **InnoDB** è un motore per il salvataggio di dati per MySQL, fornito in tutte le sue distribuzioni (Default dalla versione 5.5).
- La sua caratteristica principale è quella di supportare le transazioni di tipo **ACID**.

# Engine: InnoDB

- **InnoDB** mette a disposizione le seguenti funzionalità:
  - transazioni SQL con savepoint e transazioni XA;
  - lock a livello di record;
  - foreign key;
  - integrità referenziale;
  - colonne AUTOINCREMENT;
  - tablespace.
- 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.



## Engine: InnoDB

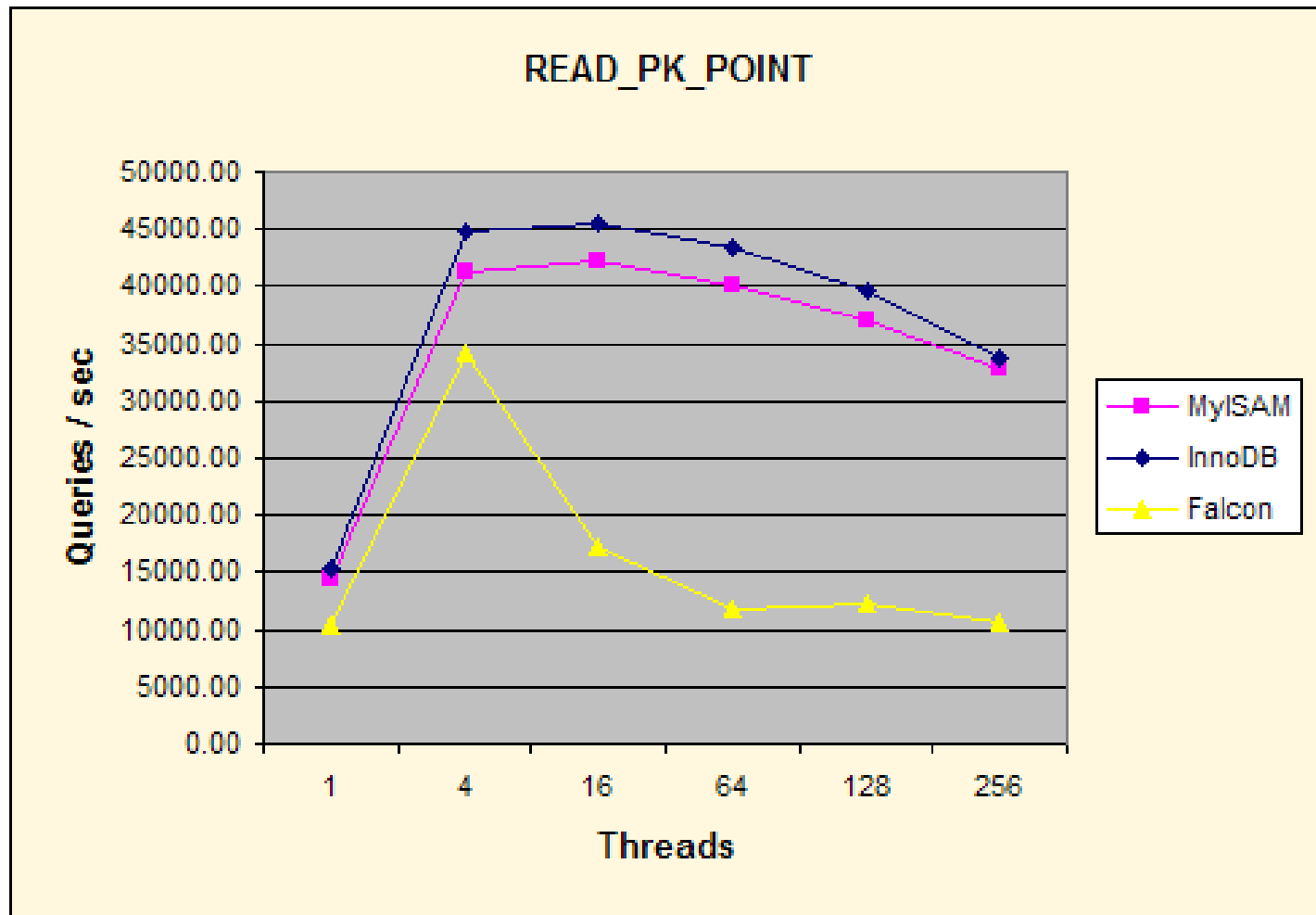
- Le tabelle InnoDB sono soggette alle seguenti limitazioni:
  - Non è possibile creare più di **1000 colonne** per tabella;
  - Su alcuni sistemi operativi le **dimensioni del tablespace** non possono superare i **2 Gb**;
  - La grandezza di tutti i **file di log** di InnoDB deve essere inferiore ai **4 Gb**;
  - La grandezza minima di un tablespace è di **10 MB**;
  - Non possono essere creati indici di tipo **FULLTEXT** con **MySQL 5.5 o precedente**;
  - Le **SELECT COUNT(\*)** su tabelle di grandi dimensioni possono essere molto lente.

# Altri tipi di Engine

```
mysql> show engines;
```

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	YES	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	DEFAULT	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

# Engine: MyISAM vs InnoDB



# Scaricare e Installare MySQL

- Alcuni link per il download di software e tool:
  - <http://dev.mysql.com/downloads/mysql/>
  - <http://dev.mysql.com/downloads/tools/>
  - <http://dev.mysql.com/downloads/connector/>
  - <http://dev.mysql.com/downloads/mysql-proxy/>
  - <https://dev.mysql.com/doc/index-other.html> (database d'esempio)
- Alternativamente si può installare un pacchetto che include web server e DBMS:
  - **EasyPhp**
  - **XAMPP**

- **mysql**: il client ufficiale per interagire con i database.
- **mysqladmin**, per lo svolgimento di ogni genere di operazione di configurazione del server;
- **mysqlcheck**, che si occupa della manutenzione delle tabelle;
- **mysqldump**, indispensabile per il backup.
- **mysqlimport**, che permette di importare tabelle nei database;
- **mysqlshow**, che fornisce informazioni su database, tabelle, indici e molto altro.

Client  
Testuali e visuali

# Connessione/disconnessione

- Per connettersi al server è necessario fornire login e password

```
shell> mysql -h host -u user -p  
Enter password: *****
```

- *host* and *user* rappresentano:
  - l'hostname dove risiede MySQL;
  - lo username di un utente che possiede un account sul server;
- *-p* specifica al server la richiesta della password all'utente.

```
C:\>mysql -u apulvirenti -p
```

```
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 46
```

```
Server version: 5.7.17 MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql>
```



# Creazione di un account

- Tipicamente viene eseguita dall'utente *root*, mediante l'uso del comando GRANT.

```
C:\>mysql -u root -p
```

```
Enter password: ****
```

```
mysql> GRANT ALL ON nomeDB.* to  
-> 'user'@'localhost' IDENTIFIED BY  
-> 'nome_password';
```

# Connessione da un client

- Per consentire la connessione da un server specifico.

```
mysql> GRANT ALL ON nomeDB.* to  
-> 'user'@'nome_server' IDENTIFIED BY  
-> 'nome_password';
```

# SHOW

- SHOW ha diverse opzioni e da informazioni riguardo ai database alle tabella, collone, indici, ecc. Da anche informazioni riguardo il server.

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']  
SHOW CREATE DATABASE db_name  
SHOW CREATE TABLE tbl_name  
SHOW DATABASES [LIKE 'pattern']  
SHOW [STORAGE] ENGINES  
SHOW ERRORS [LIMIT [offset,] row_count]  
SHOW GRANTS FOR user  
SHOW INDEX FROM tbl_name [FROM db_name]  
SHOW INNODB STATUS  
SHOW [BDB] LOGS  
SHOW PRIVILEGES  
SHOW [FULL] PROCESSLIST  
SHOW STATUS [LIKE 'pattern']  
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']  
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']  
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']  
SHOW WARNINGS [LIMIT [offset,] row_count]
```

# Tipi di Dati: Numerici

Tipo	Byte	Min. Value (Signed/Unsigned)	Max. Value (Signed/Unsigned)
TINYINT	1	-128/0	127/255
SMALLINT	2	-32.768/0	32.767/65535
MEDIUMINT	3	-8.388.608/0	8.388.607/16.777.215
INT	4	-2.147.483.648/0	2.147.483.647/4.294.967.295
BIGINT	8	-9.223.372.036.854.775.808/0	9.223.372.036.854.775.807 /18.446.744.073.709.551.615
FLOAT	4	+/-1.175494351E-38	+/-3.402823466E+38
DOUBLE	8	+/-2.225073858507201E-308	+/-1.7976931348623157E+308

# Tipi di Dati: Numerici

Tipo	Byte	Min. Value (Signed/Unsigned)	Max. Value (Signed/Unsigned)
INTEGER	Equivale ad INT		
DOUBLE PRECISION	Equivale a DOUBLE		
REAL	Equivale a DOUBLE		
DECIMAL(M[,D])	M+2	Tutti i numeri di M cifre di cui D decimali.	
NUMERIC(M[,D])	Equivale a DECIMAL		
BIT(M)	M	una sequenza di M bit (MySQL 5.5)	

# Tipi di Dati: Date e Tempi

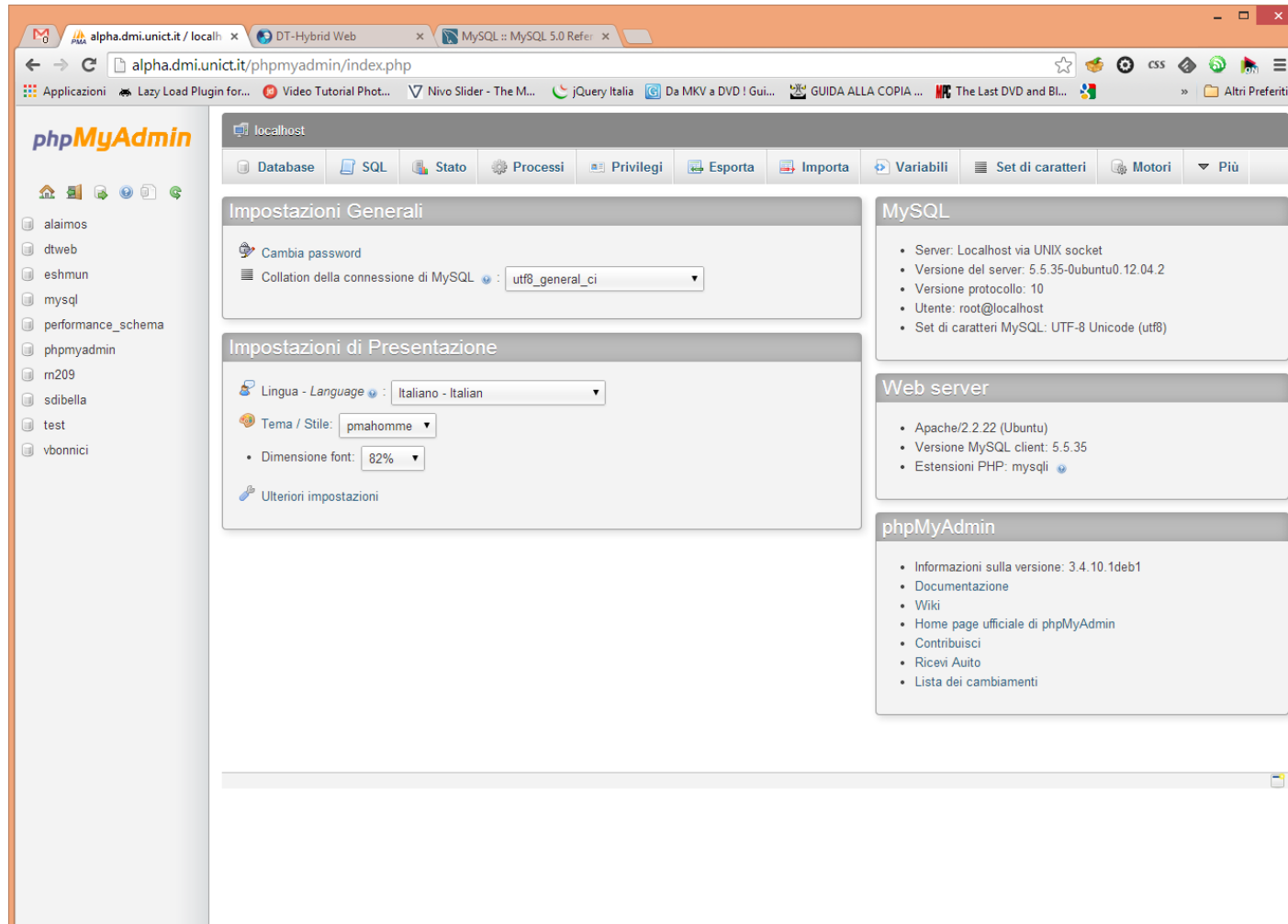
Tipo	Byte	Range
DATE	3	dal '01/01/1000' al '31/12/9999'
DATETIME	8	dal '01/01/1000 00:00:00' al '31/12/9999 23:59:59'
TIMESTAMP[(M)]	4	dal '01/01/1970' al '31/12/2037'
TIME	3	da '-838:59:59' a '838:59:59'
YEAR[(M)]	1	per YEAR(4) dal '1901' al '2144'

# Tipi di Dati: Testo e Altro

Tipo	Byte	Max Length
CHAR[(M)]/BINARY[(M)]	M	M
VARCHAR(M)/VARBINARY[(M)]	L+1	M
TINYBLOB/ TINYTEXT	L+1	255
BLOB/TEXT	L+2	65.535
MEDIUMBLOB/MEDIUMTEXT	L+3	16.777.215
LOB/TEXT	L+4	4.294.967.295
ENUM('value1','value2',...)	1 o 2 byte	65535 elementi
SET ('value1','value2',...)	1,2,3,4 o 8 byte	64 elementi
JSON	---	---

**L** è la lunghezza effettiva del testo memorizzato.

# PHPMyAdmin





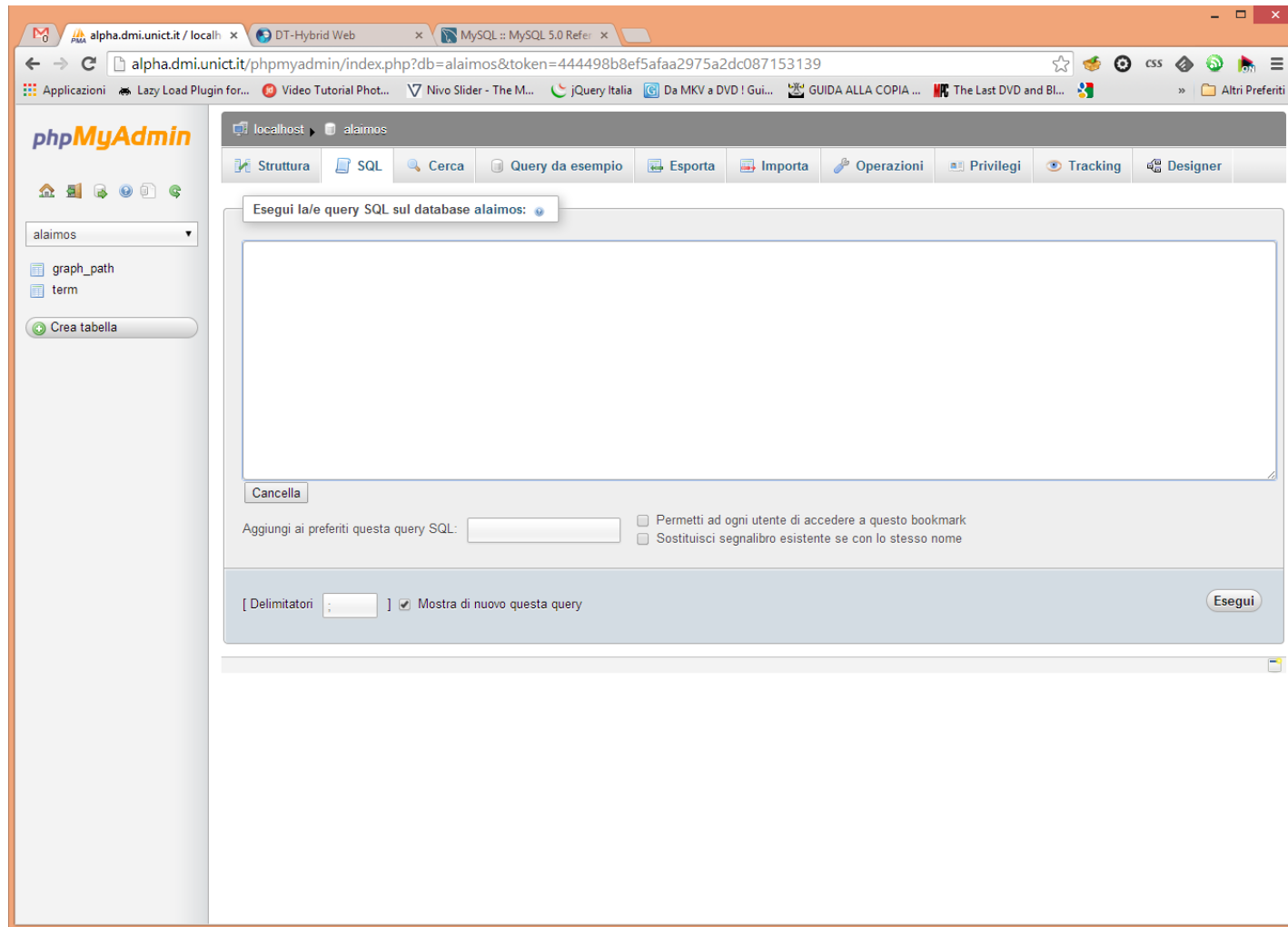
# PHPMyAdmin

The screenshot displays the PHPMyAdmin web interface in a browser window. The address bar shows the URL: `alpha.dmi.unict.it/phpmyadmin/index.php?db=alaimos&token=444498b8ef5afaa2975a2dc087153139`. The interface is in Italian and shows the 'alaimos' database selected in the left sidebar. The main panel displays a table of database statistics:

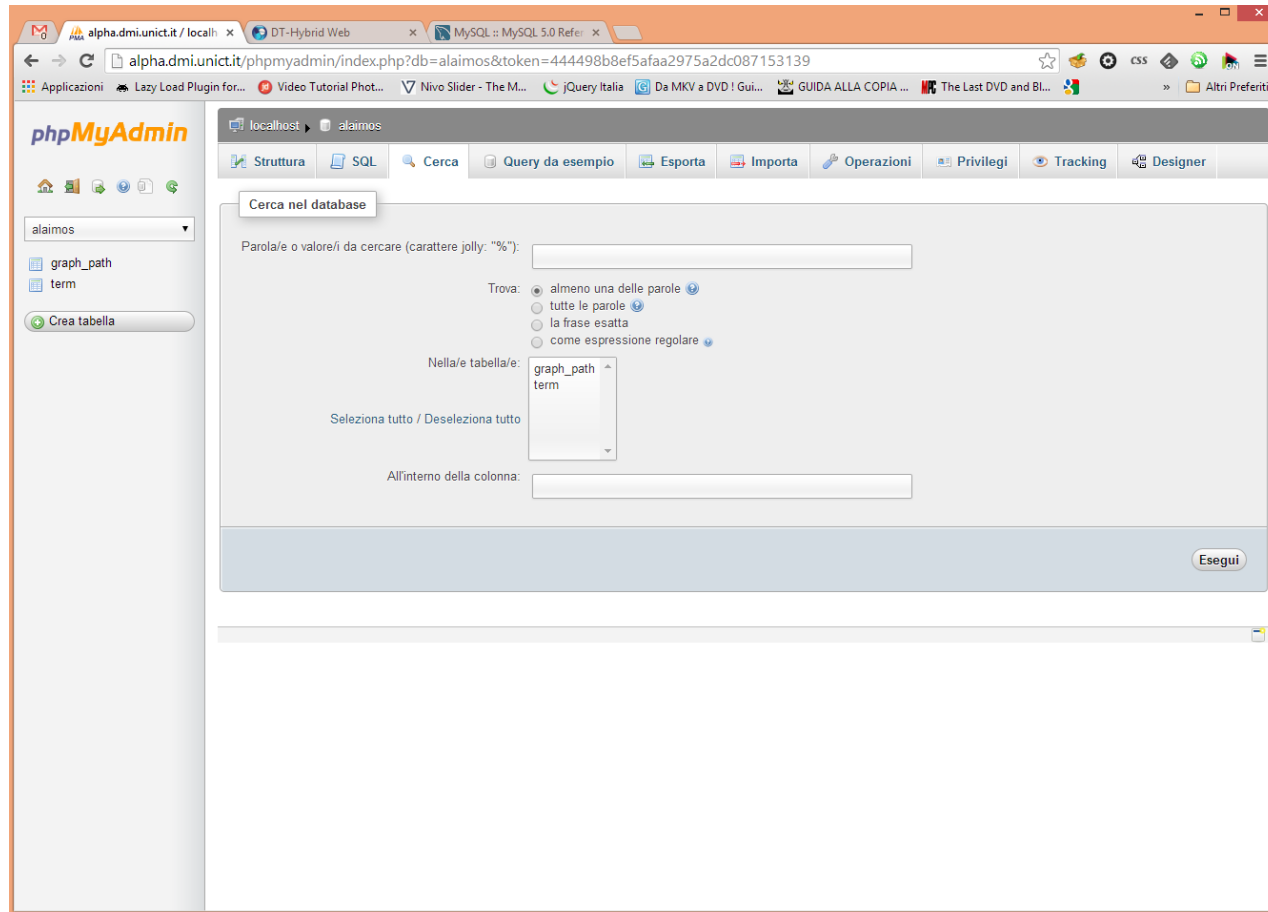
Tabella	Azione	Righe	Tipo	Collation	Dimensione	Overhead
graph_path	Mostra   Struttura   Cerca   Inserisci   Svuota   Elimina	1,043,029	MyISAM	latin1_swedish_ci	186.9 MiB	-
term	Mostra   Struttura   Cerca   Inserisci   Svuota   Elimina	40,248	MyISAM	latin1_swedish_ci	14.0 MiB	-
<b>2 tabelle</b>	<b>Totali</b>	<b>1,083,277</b>	<b>InnoDB</b>	<b>latin1_swedish_ci</b>	<b>200.9 MiB</b>	<b>0 B</b>

Below the table, there are options to 'Seleziona tutti / Deseleziona tutti' and 'Se selezionati:'. There are also links for 'Visualizza per stampa' and 'Data Dictionary'. A button 'Crea una nuova tabella nel database alaimos' is visible, followed by input fields for 'Nome:' and 'Numero di colonne:'. An 'Esegui' button is at the bottom right of the form.

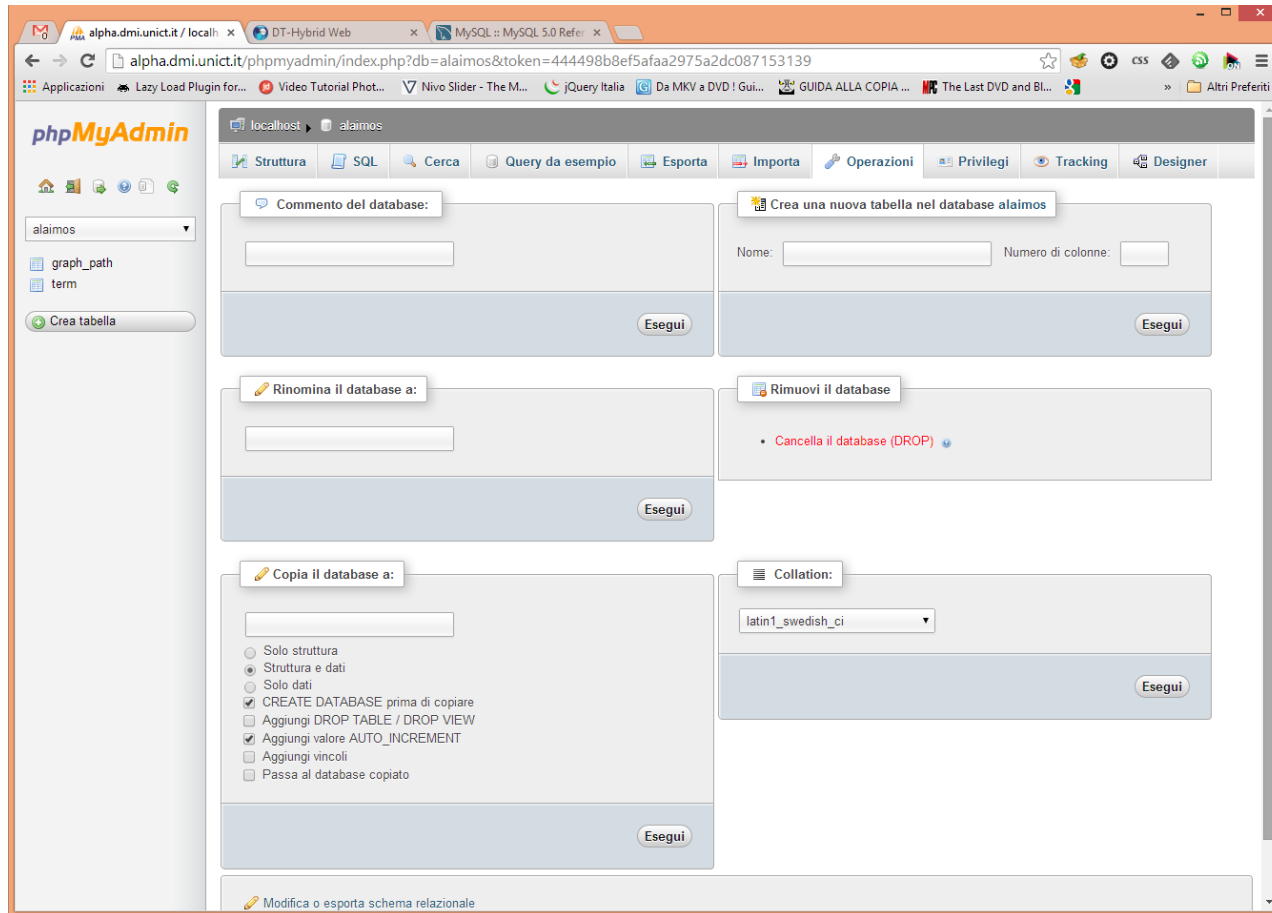
# PHPMyAdmin



# PHPMyAdmin



# PHPMyAdmin



# PHPMyAdmin

The screenshot displays the PHPMyAdmin interface in a web browser. The address bar shows the URL: `alpha.dmi.unict.it/phpmyadmin/index.php?db=alaimos&token=444498b8ef5afaa2975a2dc087153139`. The left sidebar contains the database name 'alaimos' and a list of tables: 'graph\_path' and 'term'. The main panel shows the 'graph\_path' table selected. A green status bar indicates: 'Mostrando i righi 0 - 29 ( 1,043,029 totale, La query ha impiegato 0.0270 sec)'. The SQL query entered is: 

```
SELECT * FROM 'graph_path' LIMIT 0, 30
```

. Below the query, there are options for 'Profiling [Inline]', 'Modifica', 'Spiega SQL', 'Crea il codice PHP', and 'Aggiorna'. The table display settings show 'Numero pagina: 1', 'Mostra: 30', 'righe a partire da # 30', 'in modalità orizzontale', and 'e ripeti gli headers dopo 100 celle'. The table is ordered by 'Nessuno'. The table data is as follows:

	id	term1_id	term2_id	relationship_type_id	distance	relation_distance
<input type="checkbox"/> Modifica	1	22	19	1	1	1
<input type="checkbox"/> Modifica	2	22	21	1	1	1
<input type="checkbox"/> Modifica	3	25	16058	20	1	1
<input type="checkbox"/> Modifica	4	25	17957	1	1	1
<input type="checkbox"/> Modifica	5	26	13387	1	1	1
<input type="checkbox"/> Modifica	6	26	13388	1	1	1
<input type="checkbox"/> Modifica	7	26	13999	20	1	1
<input type="checkbox"/> Modifica	8	26	16512	1	1	1
<input type="checkbox"/> Modifica	9	26	16513	1	1	1
<input type="checkbox"/> Modifica	10	26	26807	20	1	1
<input type="checkbox"/> Modifica	11	29	28	1	1	1
<input type="checkbox"/> Modifica	12	29	35	1	1	1
<input type="checkbox"/> Modifica	13	29	69	1	1	1
<input type="checkbox"/> Modifica	14	29	78	1	1	1
<input type="checkbox"/> Modifica	15	29	234	1	1	1
<input type="checkbox"/> Modifica	16	29	285	1	1	1
<input type="checkbox"/> Modifica	17	29	289	1	1	1
<input type="checkbox"/> Modifica	18	29	292	1	1	1

# PHPMyAdmin

The screenshot displays the PHPMyAdmin interface in a web browser. The address bar shows the URL: `alpha.dmi.unict.it/phpmyadmin/index.php?db=alaimos&token=444498b8ef5afaa2975a2dc087153139`. The interface is in Italian and shows the 'term' table structure for the 'alaimos' database.

**Table Structure (Struttura):**

#	Campo	Tipo	Collation	Attributi	Nulli	Predefinito	Extra	Azione
1	id	int(11)			No	nessuno	AUTO_INCREMENT	Modifica Elimina Più
2	name	varchar(255)	latin1_swedish_ci		No	nessuno		Modifica Elimina Più
3	term_type	varchar(255)	latin1_swedish_ci		No	nessuno		Modifica Elimina Più
4	acc	varchar(255)	latin1_swedish_ci		No	nessuno		Modifica Elimina Più
5	is_obsolete	int(11)			No	0		Modifica Elimina Più
6	is_root	int(11)			No	0		Modifica Elimina Più
7	is_relation	int(11)			No	0		Modifica Elimina Più

**Indici (Indexes):**

Azione	Chiave	Tipo	Unica	Compresso	Campo	Cardinalità	Collation	Nulli	Commenti
Modifica Elimina	PRIMARY	BTREE	Sì	No	id	40248	A		
Modifica Elimina	acc	BTREE	Sì	No	acc	40248	A		
Modifica Elimina	t1	BTREE	No	No	name	40248	A		
Modifica Elimina	t2	BTREE	No	No	term_type	9	A		
Modifica Elimina	t4	BTREE	No	No	id	40248	A		
Modifica Elimina	t5	BTREE	No	No	acc	40248	A		
Modifica Elimina	t6	BTREE	No	No	id	40248	A		
Modifica Elimina	t6	BTREE	No	No	name	40248	A		
Modifica Elimina	t6	BTREE	No	No	term_type	40248	A		

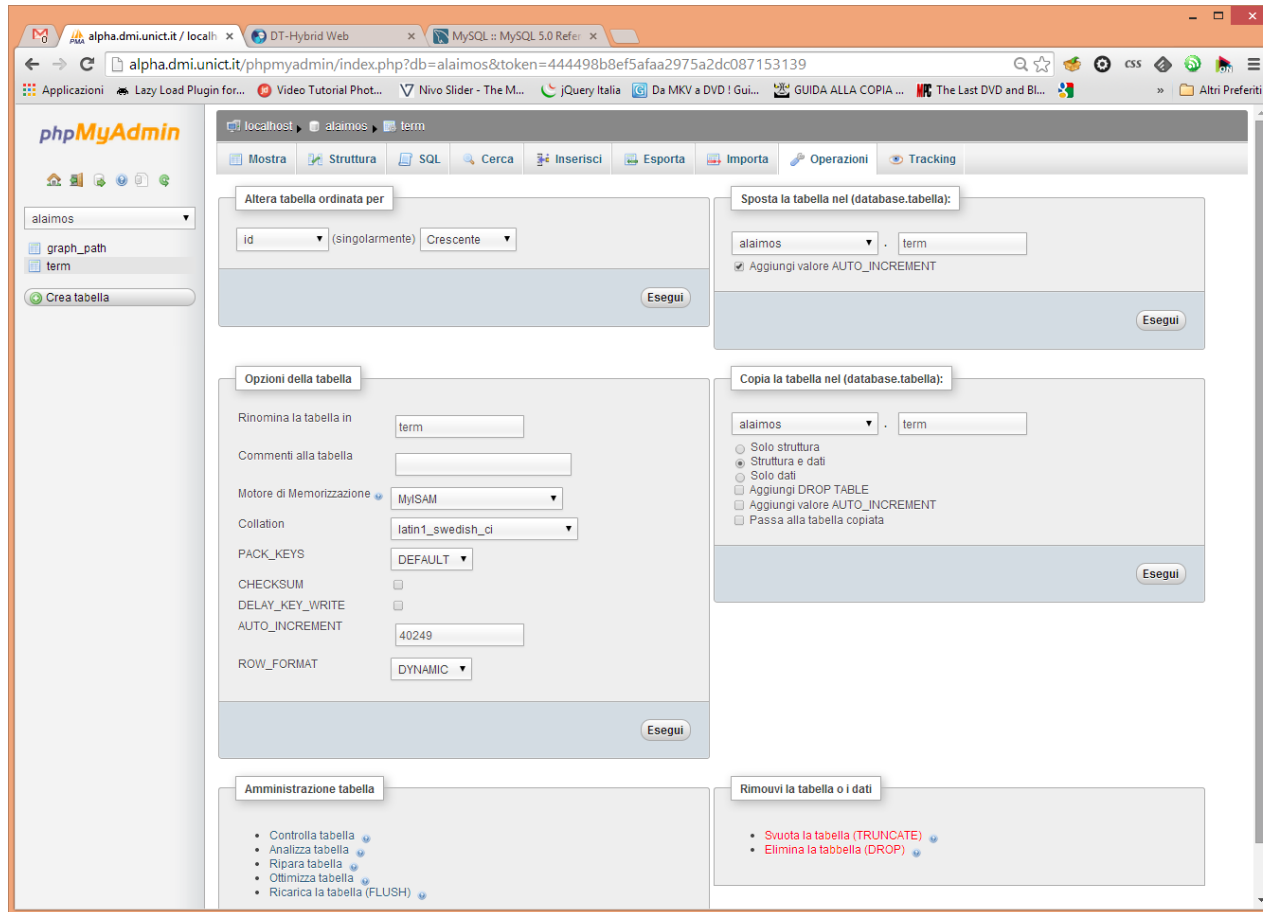
**Spazio utilizzato (Space used):**

Tipo	Utilizzo
Dati	3,227.0 KiB
Indice	7,095.0 KiB
Totale	10,322.0 KiB

**Statistiche righe (Row statistics):**

Istruzioni	Valore
Formato	dynamic
Collation	latin1_swedish_ci
Righe	40,248
Lunghezza riga	82
Dimensione riga	263 B
Prossimo Autoindex:	40,249
Creazione	Ago 15, 2014 alle 13:30
Ultimo cambiamento	Ago 15, 2014 alle 13:30
Ultimo controllo	Ago 15, 2014 alle 13:30

# PHPMyAdmin



# PHPMyAdmin

The screenshot shows the PHPMyAdmin interface in a web browser. The browser's address bar displays the URL: `alpha.dmi.unict.it/phpmyadmin/index.php?db=alaimos&token=444498b8ef5afaa2975a2dc087153139`. The interface is for the 'alaimos' database. On the left sidebar, there are navigation icons and a list of tables: 'graph\_path' and 'term'. A 'Crea tabella' (Create table) button is visible. The main area is titled 'Struttura' (Structure) and contains a form for creating a new table. The form includes fields for 'Nome tabella:' (Table name), 'Campo' (Field), 'Tipo' (Type), 'Lunghezza/Set\*' (Length/Set), 'Predefinito' (Default), 'Collation', 'Attributi' (Attributes), 'Null', 'Indice' (Index), 'AUTO\_INCREMENT', 'Commenti' (Comments), 'Tipo MIME', 'Trasformazione del browser' (Browser transformation), and 'Opzioni di trasformazione' (Transformation options). At the bottom, there are fields for 'Commenti alla tabella:' (Table comments), 'Motore di Memorizzazione:' (Storage engine), 'Collation:', and 'Definizione PARTITION:'. A 'Salva' (Save) button is at the bottom right, along with a checkbox for 'Oppure Aggiungi' (Or Add) and a text input for the number of fields (currently '1').

Nome tabella:

Struttura

Campo

Tipo

Lunghezza/Set\*

Predefinito

Collation

Attributi

Null

Indice

AUTO\_INCREMENT

Commenti

Tipo MIME

Trasformazione del browser

Opzioni di trasformazione

Commenti alla tabella:

Motore di Memorizzazione:

Collation:

Definizione PARTITION:

Salva Oppure Aggiungi 1 campi Esegui



Account e Privilegi

# Account e Privilegi

- Come creare un utente:

**CREATE USER** *'name'@'host'* **IDENTIFIED BY** *'PASSWORD'*;

- Come assegnare privilegi ad un utente:

**GRANT** *privilege,...* **ON** *\*| 'db'. \*| 'db'. 'table'* **TO** *'username'@'host', ...;*

- **'privilege'** può essere uno dei seguenti valori:
  - **ALL**
  - **USAGE**
  - **SELECT, INSERT, UPDATE, DELETE**
  - **CREATE, ALTER, INDEX, DROP, CREATE VIEW, TRIGGER**

# Account e Privilegi

- Rimuovere un utente:

**DROP USER** *'name'@'host',...;*

- Rimuovere i privilegi di un utente:

**REVOKE** *privilege,... ON \*| 'db'. \*| 'db'. 'table' FROM 'username'@'host', ...;*

- **'privilege'** può essere uno dei seguenti valori:
  - ***ALL PRIVILEGES***
  - ***USAGE***
  - ***SELECT, INSERT, UPDATE, DELETE***
  - ***CREATE, ALTER, INDEX, DROP, CREATE VIEW, TRIGGER***

Comandi

# Primi Esempi

— 1. Mostra il nome dell'utente corrente

```
SELECT user();
```

— 2. Mostra la versione, la data, e l'ora corrente

```
SELECT version(), current_date, current_time, current_timestamp;
```

— 3. Mostra l'elenco dei database

```
SHOW DATABASES;
```

# Gestione Database

- Creazione di un database:

**CREATE DATABASE [IF NOT EXISTS]** *nome*;

- Cancellazione di un database:

**DROP DATABASE [IF EXISTS]** *nome*;

- Accesso ad un database:

**USE** *nome*;

# Gestione Tabelle

```
CREATE TABLE [IF NOT EXISTS] nome (  
    campo1 TIPO1 ALTRI PARAMETRI,  
    campo2 TIPO2 ALTRI PARAMETRI,  
    ...  
    campoN TIPON ALTRI PARAMETRI,  
    PRIMARY KEY(campo1, campo2)  
) ENGINE=InnoDB;
```

- **IF NOT EXISTS**: crea la tabella solo se non esiste.
- Altri parametri:
  - **NOT NULL**: non permette valori NULL nella colonna;
  - **AUTO\_INCREMENT**: una colonna il cui valore è calcolato automaticamente in base ad un contatore interno;
  - **DEFAULT** *valore*: specifica un valore di default per un campo;
  - **NULL**: specifica che un campo può contenere valori NULL.

# Gestione Database e Tabelle

```
-- 1. Crea un database
CREATE DATABASE IF NOT EXISTS prova;

-- 2. Seleziona il database di prova
USE prova;

-- 3. Crea una tabella di esempio
CREATE TABLE country (
    country_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    country VARCHAR(50) NOT NULL,
    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (country_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE city (
    city_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    city VARCHAR(50) NOT NULL,
    country_id SMALLINT UNSIGNED NOT NULL,
    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (city_id),
    KEY idx_fk_country_id (country_id),
    CONSTRAINT `fk_city_country` FOREIGN KEY (country_id) REFERENCES country (country_id) ON DELETE RESTRICT ON UPDATE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
1  [CONSTRAINT [symbol]] FOREIGN KEY
2      [index_name] (col_name, ...)
3      REFERENCES tbl_name (col_name,...)
4      [ON DELETE reference_option]
5      [ON UPDATE reference_option]
6
7  reference_option:
8      RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```



# Gestione Tabelle

**DESCRIBE** *nome*: mostra informazioni sui campi contenuti in una tabella.

**INSERT INTO** *table* (*field1*, ..., *fieldN*) **VALUES** (*value1*, ..., *valueN*) [ **ON DUPLICATE KEY UPDATE**

*field1=**value1*,

...

*fieldN=**valueN* ]

**INSERT INTO** *table* (*field1*, ..., *fieldN*) **SELECT** ... [ **ON DUPLICATE KEY UPDATE**

*field1=**value1*,

...

*fieldN=**valueN* ]

# Gestione Database e Tabelle

```
-- 4. Uso del comando DESCRIBE
DESCRIBE city;

-- 5. Esempio Insert
INSERT INTO country (country_id, country) VALUES (1, 'Afghanistan'), (NULL, 'Algeria');
INSERT INTO city (city_id, city, country_id) VALUES (1, 'Kabul', 1);

-- 6. Cancella la tabella di esempio
DROP TABLE city;

-- 7. Cancella un database
DROP DATABASE IF EXISTS prova;
```

Field	Type	Null	Key	Default	Extra
city_id	smallint(5) unsigned	NO	PRI	NULL	auto_increment
city	varchar(50)	NO		NULL	
country_id	smallint(5) unsigned	NO	MUL	NULL	
last_update	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP

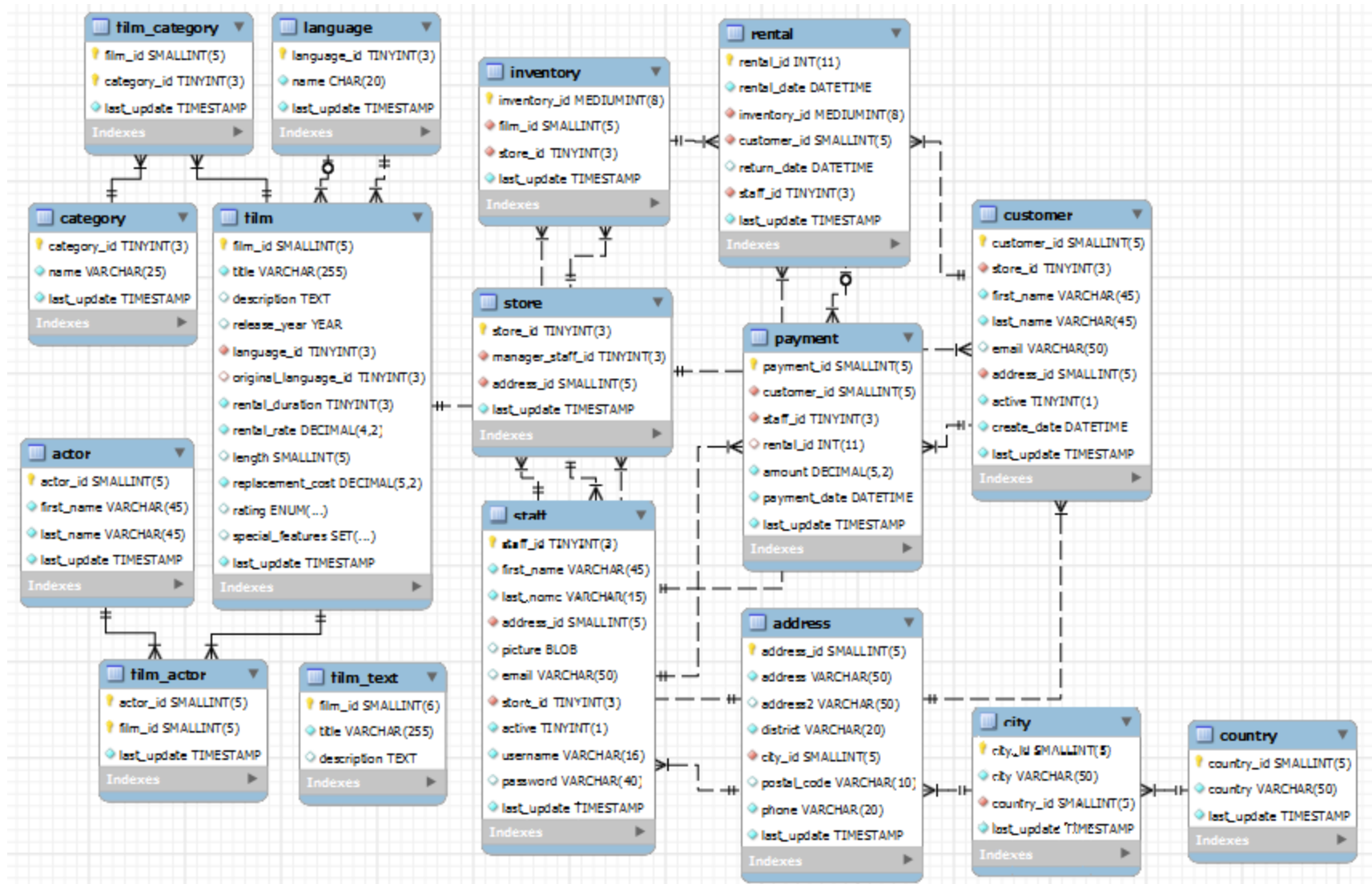
## SELECT

```
[ALL | DISTINCT | DISTINCTROW ]
select_expr [, select_expr] ...
[into_option]
[FROM table_references
  [PARTITION partition_list]]
[WHERE where_condition]
[GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position}
  [ASC | DESC], ... [WITH ROLLUP]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[into_option]
[FOR {UPDATE | SHARE}
  [OF tbl_name [, tbl_name] ...]
  [NOWAIT | SKIP LOCKED]
  | LOCK IN SHARE MODE]
[into_option]
```

```
into_option: {
  INTO OUTFILE 'file_name'
    [CHARACTER SET charset_name]
    export_options
| INTO DUMPFILE 'file_name'
| INTO var_name [, var_name] ...
}
```

## Select

# Sakila Databases



# Select

```
-- 1. Seleziona gli identificativi, i nomi e le email dei primi 10 clienti ordinati per cognome e nome
SELECT customer_id, last_name, first_name, email
      FROM customer
      ORDER BY last_name ASC, first_name ASC
      LIMIT 0,10;

-- 2. Seleziona gli utenti che sono stati aggiunti dopo le 22:04:37 del 14 aprile 2006
SELECT * FROM customer
      WHERE create_date >= '2006-02-14 22:04:37'
      ORDER BY last_name ASC;

-- 3. Cerca i film il cui nome inizia per "W"
SELECT * FROM film
      WHERE title LIKE 'W%'
      ORDER BY title ASC;

-- 4. Altro modo
SELECT * FROM film
      WHERE title REGEXP '^W'
      ORDER BY title ASC;

-- 5. Cerca i film che contengono "W*R"
SELECT * FROM film
      WHERE title LIKE '%W_R%'
      ORDER BY title ASC;
```

# I valori NULL

- Il valore **NULL** per un campo assume il significato di «mancante, sconosciuto»
  - Esso è trattato diversamente dagli altri valori.
- Per testare il valore di **NULL** non si possono usare i consueti operatori di confronto =, <, o <>
  - Esistono due operatori di confronto appositi per valori **NULL**:
    - **IS NULL**
    - **IS NOT NULL**
- Quando si usa **ORDER BY** i valori **NULL** sono inseriti all'inizio con **ASC** ed alla fine con **DESC**.

# I Valori NULL

-- 6. Cerca i film che non sono ancora stati restituiti;

```
SELECT F.film_id, F.title, F.description, I.store_id, R.return_date
FROM film AS F, inventory AS I, rental AS R
WHERE
    I.film_id = F.film_id AND
    R.inventory_id = I.inventory_id AND
    R.return_date = NULL
ORDER BY F.title ASC;
```

-- 7. Cerca i film che non sono ancora stati restituiti;

```
SELECT F.film_id, F.title, F.description, I.store_id, R.return_date
FROM film AS F, inventory AS I, rental AS R
WHERE
    I.film_id = F.film_id AND
    R.inventory_id = I.inventory_id AND
    R.return_date IS NULL
ORDER BY F.title ASC;
```

# Select

-- 8. Quante volte un film è stato noleggiato e restituito?

```
SELECT F.film_id, F.title, COUNT(R.rental_id) AS Conteggio
FROM film AS F, inventory AS I, rental AS R
WHERE
    I.film_id = F.film_id AND
    R.inventory_id = I.inventory_id
GROUP BY F.film_id
ORDER BY Conteggio DESC;
```

-- 9. Perché non compaiono quelli con 0 noleggi e restituzioni? Bisogna costruire una query più complessa

```
SELECT F.film_id, F.title, IFNULL(C.Conteggio, 0) AS ConteggioCorretto
FROM film AS F
LEFT JOIN (
    SELECT I.film_id, COUNT(R.rental_id) AS Conteggio
    FROM inventory AS I
    LEFT JOIN rental AS R ON R.inventory_id = I.inventory_id
    GROUP BY I.film_id
) AS C ON F.film_id = C.film_id
ORDER BY Conteggio DESC;
```



# Update

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
  SET col_name1={expr1!DEFAULT} [, col_name2={expr2!DEFAULT}] ...
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
  SET col_name1={expr1!DEFAULT} [, col_name2={expr2!DEFAULT}] ...
  [WHERE where_condition]
```

- **SET**: specifichiamo quali colonne modificare e quali valori assegnare;
- **WHERE**: le condizioni che determinano quali righe saranno modificate;
- **ORDER BY**: per decidere in che ordine effettuare gli aggiornamenti;
- **LIMIT**: per indicare il numero massimo di righe da modificare.

# Delete

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
FROM table_references
    [WHERE where_condition]
```

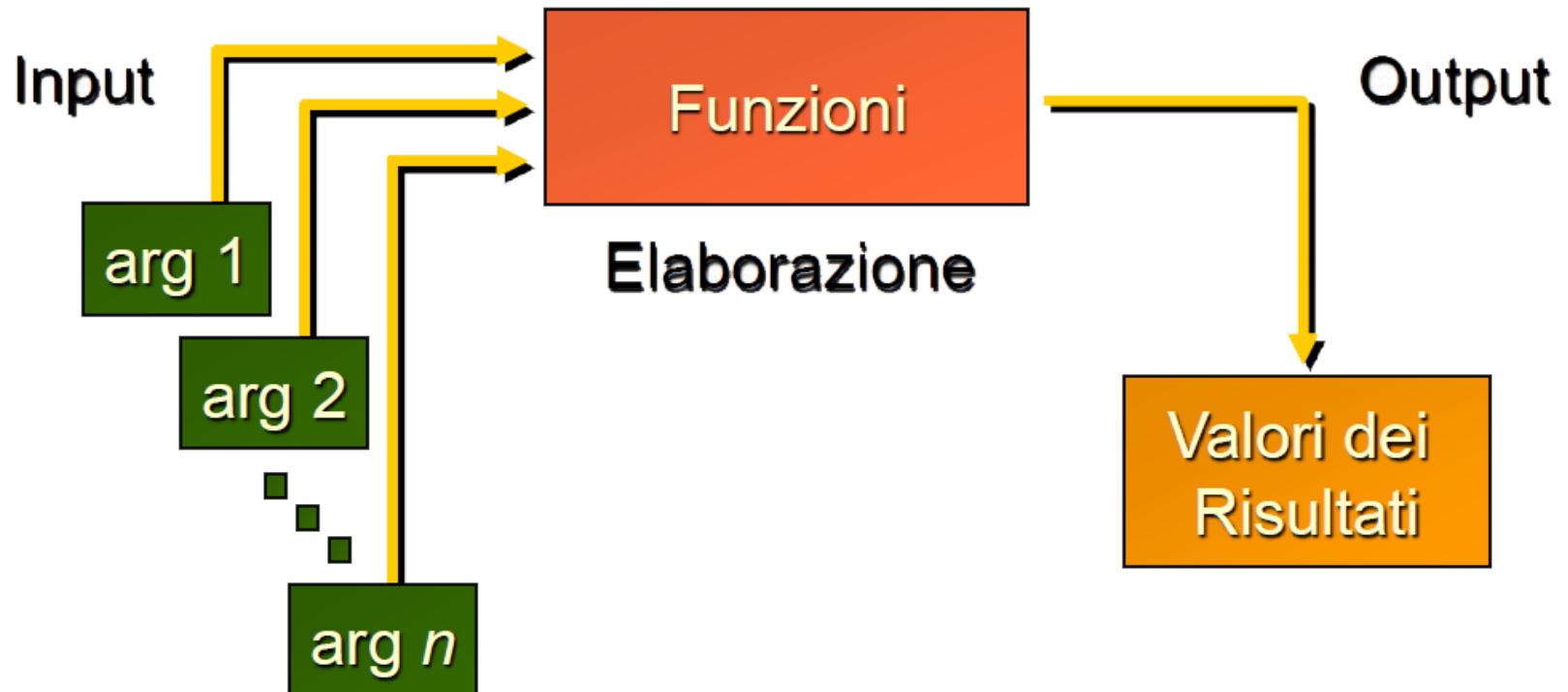
Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

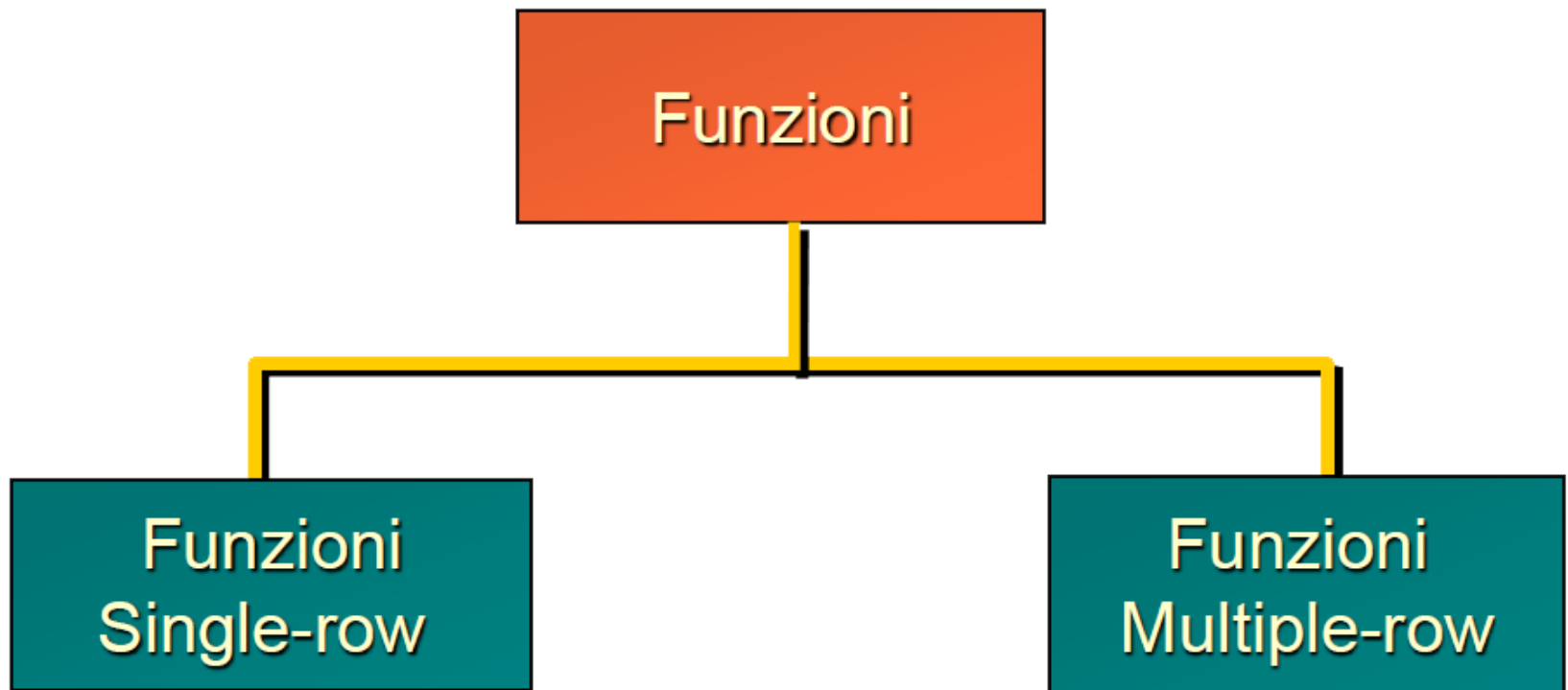
- **ORDER BY** e **LIMIT**: funzionano come in **UPDATE**;
- **WHERE**: stabilisce le condizioni in base alle quali le righe verranno eliminate .

Funzioni e operatori

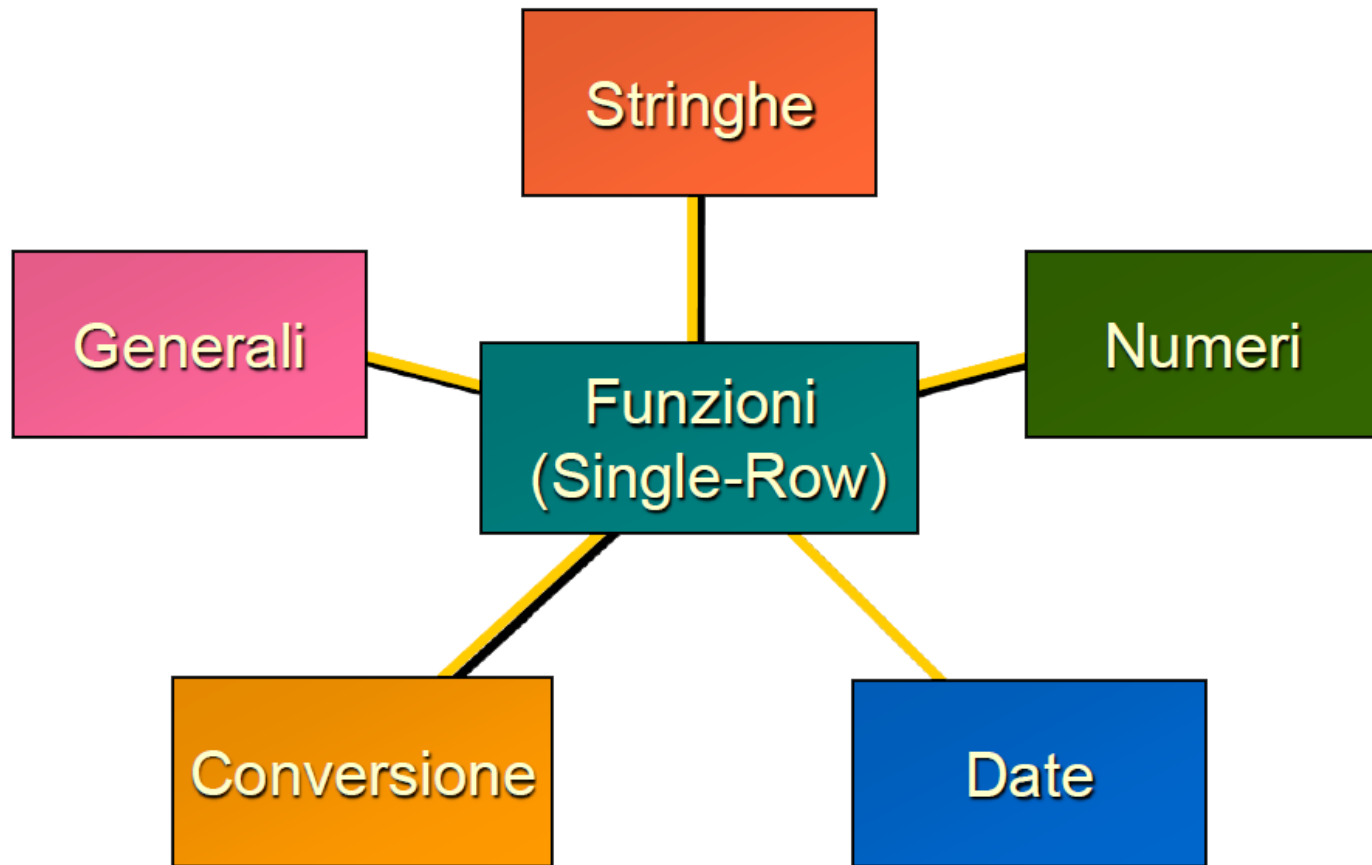
# Funzioni



# Funzioni



# Funzioni



# Operatori

- Aritmetici:
  - "+" (addizione);
  - "-" (sottrazione);
  - "\*" (moltiplicazione);
  - "/" (divisione);
  - "%" (modulo);
- Matematici:
  - **ABS(X)**
  - **FLOOR(X)**
  - **CEILING(X)**
  - **SIN(X)**
  - **COS(X)**
  - **LN(X)**
  - **LOG(X)**
  - **LOG(B,X)**
- Logici:
  - **NOT(!)**;
  - **AND(&&)**;
  - **OR(||)**
  - **XOR**;
- Confronto (Risultato 1 o 0):
  - **=, <=>** (NULL-safe);
  - **<>, !=**;
  - **<=, <, >=** ;
  - **>**;
  - **IS NULL**;
  - **IS NOT NULL**.

# Operatori

- Per controllare se un numero è all'interno di un range di valori si può usare una delle seguenti espressioni:
  - *expr* **BETWEEN** *min* **AND** *max*
  - *expr* **NOT BETWEEN** *min* **AND** *max*
- Per confrontare rispetto ad una lista fissata di valori:
  - *expr* **IN** (*value*, ...)
  - *expr* **NOT IN** (*value*, ...)
- **COALESCE**(*val*, ...): restituisce il primo elemento non-NULL di una lista;
- **INTERVAL**(*N*,*N1*,*N2*,*N3*,...):  
Ritorna:
  - 0 se *N* < *N1*;
  - 1 se *N* < *N2*;
  - ecc...
  - -1 se *N* è NULL.



# Controllo del flusso

```
CASE value
    WHEN compare_value THEN result
    [WHEN compare_value THEN result ...]
    [ELSE result]
END
```

```
CASE
    WHEN condition THEN result
    [WHEN condition THEN result ...]
    [ELSE result]
END
```

```
IF (expr1, expr2, expr3)
```

```
IFNULL (expr1, expr2)
```

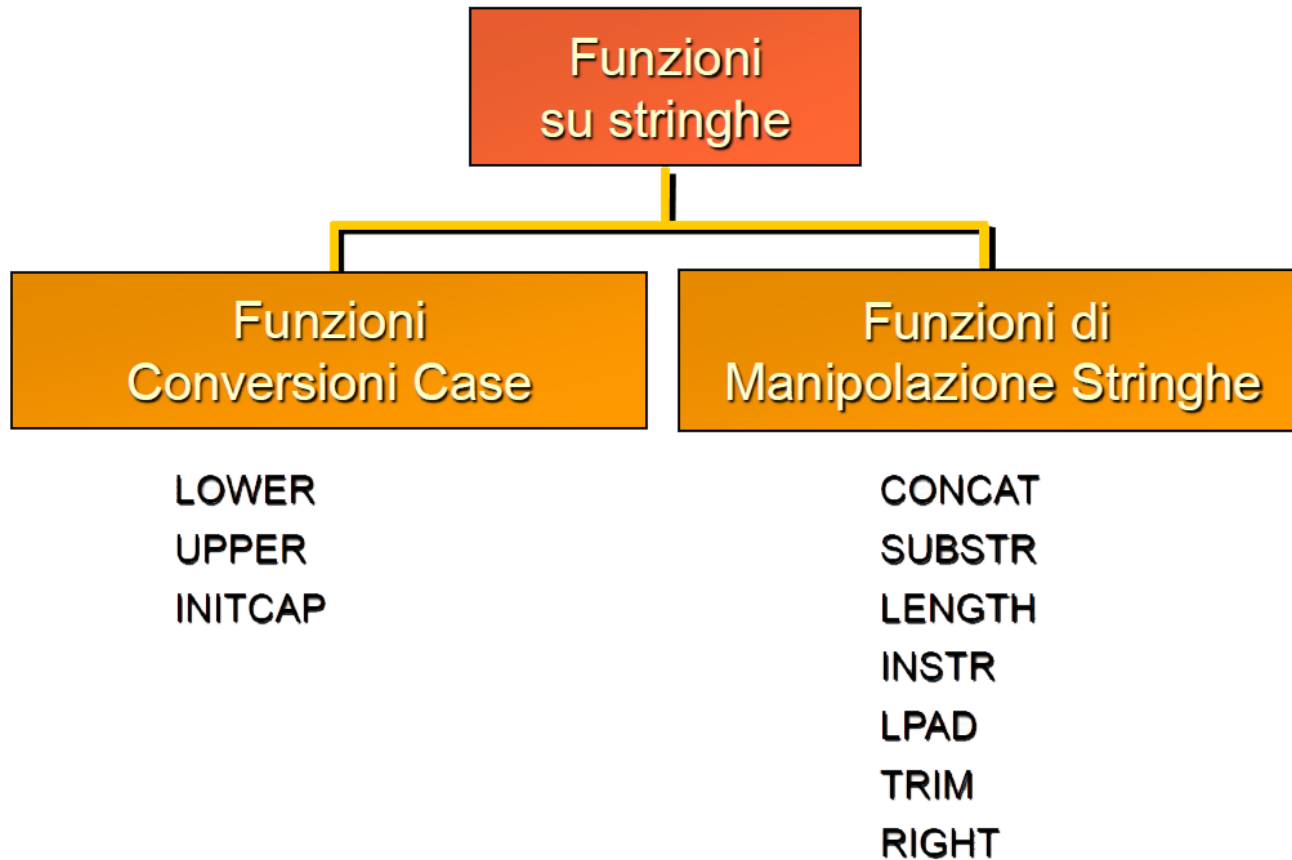
```
NULLIF (expr1, expr2)
```

# Controllo del flusso

```

.....
SELECT film_id, title,
       CASE rating
       WHEN 'G' THEN 'General Audiences'
       WHEN 'PG' THEN 'Parental Guidance Suggested'
       WHEN 'PG-13' THEN 'Parents Strongly Cautioned'
       WHEN 'R' THEN 'Restricted'
       WHEN 'NC-17' THEN 'No one 17 and under admitted'
       ELSE 'There is nothing else'
       END AS ExplainedRating,
       IF(STRCMP(rating, 'NC-17')=0, 'YES', 'no') AS RequireID FROM film;
```

# Funzioni su stringhe



# Funzioni su stringhe

- Funzione conversioni case:
  - **LOWER**(*str*) , **LCASE**(*str*)
  - **UPPER**(*str*) , **UCASE**(*str*)
- Funzioni manipolazione stringhe:
  - **ASCII**(*str*): ritorna il valore numerico del carattere più a sinistra di *str*;
  - **BIN**(*N*): Ritorna una stringa che rappresenta il valore binario di *N*;
  - **CONCAT\_WS**(*separator, str1, str2, ...*): il primo argomento è il separatore il resto gli argomenti;
  - **CONV**(*N, from\_base, to\_base*): converte i numeri tra differenti basi;
  - **BIT\_LENGTH**(*str*): Ritorna la lunghezza della stringa *str* in bit;
  - **CHAR**(*N, ...*): interpreta ogni argomento *N* come intero e ritorna una stringa consistente dei caratteri dati dal codice numerico degli interi;
  - **CHAR\_LENGTH**(*str*): ritorna la lunghezza della stringa misurata in caratteri;
  - **CONCAT**(*str1, str2, ...*): ritorna la stringa che si ottiene concatenando gli argomenti. Ritorna NULL se un argomento è NULL.

# Funzioni su stringhe

- Funzioni manipolazione stringhe:
  - **ELT**(*N, str1, str2, str3, ...*): Ritorna str1 se N=1, str2 se N=2 ect...;
  - **FIELD**(*str, str1, str2, str3, ...*): Ritorna la posizione di str in str1, str2, ect...;
  - **FIND\_IN\_SET**(*str, strlist*): Ritorna un valore nel range tra 1 a N se la stringa str è nella lista delle stringhe strlist (ovvero N sottostringhe, separate da “,”);
  - **HEX**(*N\_o\_S*): ritorna il valore esadecimale della stringa;
  - **LEFT**(*str, len*): Ritorna i len caratteri più a sinistra di str;
  - **INSTR**(*str, substr*): Ritorna la posizione della prima occorrenza della substr in str;
  - **LENGTH**(*str*): Ritorna la lunghezza della stringa str in bytes;
  - **LOCATE**(*substr, str*) / **LOCATE**(*substr, str, pos*): La prima sintassi ritorna la posizione delle prima occorrenza di substr in str, la seconda inizia la ricerca dalla posizione pos;
  - **LTRIM**(*str*): Ritorna str con gli spazi iniziali rimossi.

# Funzioni su stringhe

- **REPEAT**(*str,count*): Ritorna una stringa formata da *str* ripetuta *count* volte;
- **REPLACE**(*str,from,to*): Rimpiazza tutte le occorrenze di *from* in *to* da *str*;
- **REVERSE**(*str*): Ritorna la stringa invertita;
- **RIGHT**(*str,len*): Ritorna i *len* caratteri più a destra di *str*;
- **RTRIM**(*str*): Ritorna la stringa *str* con gli spazi finali rimossi;
- **STRCMP**(*expr1,expr2*): Ritorna **0** (**zero**) se le due stringhe sono uguali, **-1** se il primo argomento è più piccolo del secondo, **1** altrimenti.

# Funzioni su stringhe

```
SELECT C.customer_id,  
       CONCAT(C.first_name, ' ', C.last_name) AS name,  
       CONCAT_WS(' ', A.address, IF(STRCMP(A.address2,'')=0, NULL, A.address2)) AS address,  
       A.postal_code, A.phone, CI.city, CO.country  
FROM  
       customer AS C  
       JOIN address AS A ON A.address_id = C.address_id  
       JOIN city AS CI ON CI.city_id = A.city_id  
       JOIN country AS CO ON CO.country_id = CI.country_id  
WHERE active <> 0;
```

# Ricerca Full-Text

- **MATCH** (*col1,col2,...*) **AGAINST** (*expr* [**IN BOOLEAN MODE** | **WITH QUERY EXPANSION**]):
  - **MATCH ... AGAINST** è utilizzata per ricerche full text:
    - ritorna la rilevanza tra il testo che si trova nelle colonne (*col1,col2,...*) e la query *expr*.
    - La similarità è un valore positivo in virgola mobile.
- La funzione **match** esegue una ricerca in linguaggio naturale per una stringa contro un testo che è rappresentato da una o più colonne incluse in un indice FULL TEXT.
- La ricerca di default è case-insensitive.



# Ricerca Full-Text

```
SET @WORD = 'circus';
SELECT film_id, title,
       MATCH(title, description) AGAINST (@WORD) AS score
FROM film_text
WHERE
       MATCH(title, description) AGAINST (@WORD);
```

## Ricerca Full-Text

- È possibile eseguire una ricerca full-text in boolean mode, i segni + e – indicano le parole che devono essere presenti o assenti, rispettivamente per un match che occorre.
- A volte la stringa di ricerca è troppo corta e potrebbe tralasciare dei risultati significativi, è possibile utilizzare **query expansion** per ovviare a questo problema.

# Ricerca Full-Text

```
SET @WORD = 'circus';
SELECT film_id, title,
       MATCH(title, description) AGAINST (@WORD WITH QUERY EXPANSION) AS score,
       MATCH(title, description) AGAINST (@WORD) AS score2
FROM film_text
WHERE MATCH(title, description) AGAINST (@WORD WITH QUERY EXPANSION);
```

# Stored Procedure

Prof. Alfredo Pulvirenti

Prof. Salvatore Alaimo

# Stored procedure

Le Stored Procedures sono dei programmi, scritti generalmente in linguaggio SQL, memorizzati nei database e invocati su esplicita richiesta da parte degli utenti.

Per la creazione, alcuni database managers impiegano linguaggi procedurali:

- [PL/pgSQL](#) di PostgreSQL
- [PL/SQL](#) di Oracle
- [SQL PL](#) di DB2
- [Transact-SQL](#) di SQL Server
- [MySql Stored Procedure](#) di MySQL,

# Stored procedure

Essendo SQL è un linguaggio dichiarativo, le Stored Procedures rappresentano una sua estensione procedurale grazie ai suoi costrutti:

*BEGIN, END, DECLARE, FOR, WHILE, LOOP, IF, etc*

Se scritte in altri linguaggi standard (C/C++), esse sono compilate come oggetti esterni ed integrati in MySQL.

# Stored procedure

Sono suddivise in due gruppi di sotto-programmi dotati di caratteristiche differenti:

- **Procedure:** Accetta parametri di input e non restituisce valori. L'unico modo per farlo è attraverso una variabile di output passata in input per riferimento.
- - **Funzioni (UDF):** Restituiscono un valore e accettano parametri di input ed output

I DBMS che supportano le stored procedure effettuano la loro compilazione una sola volta (al loro inserimento).

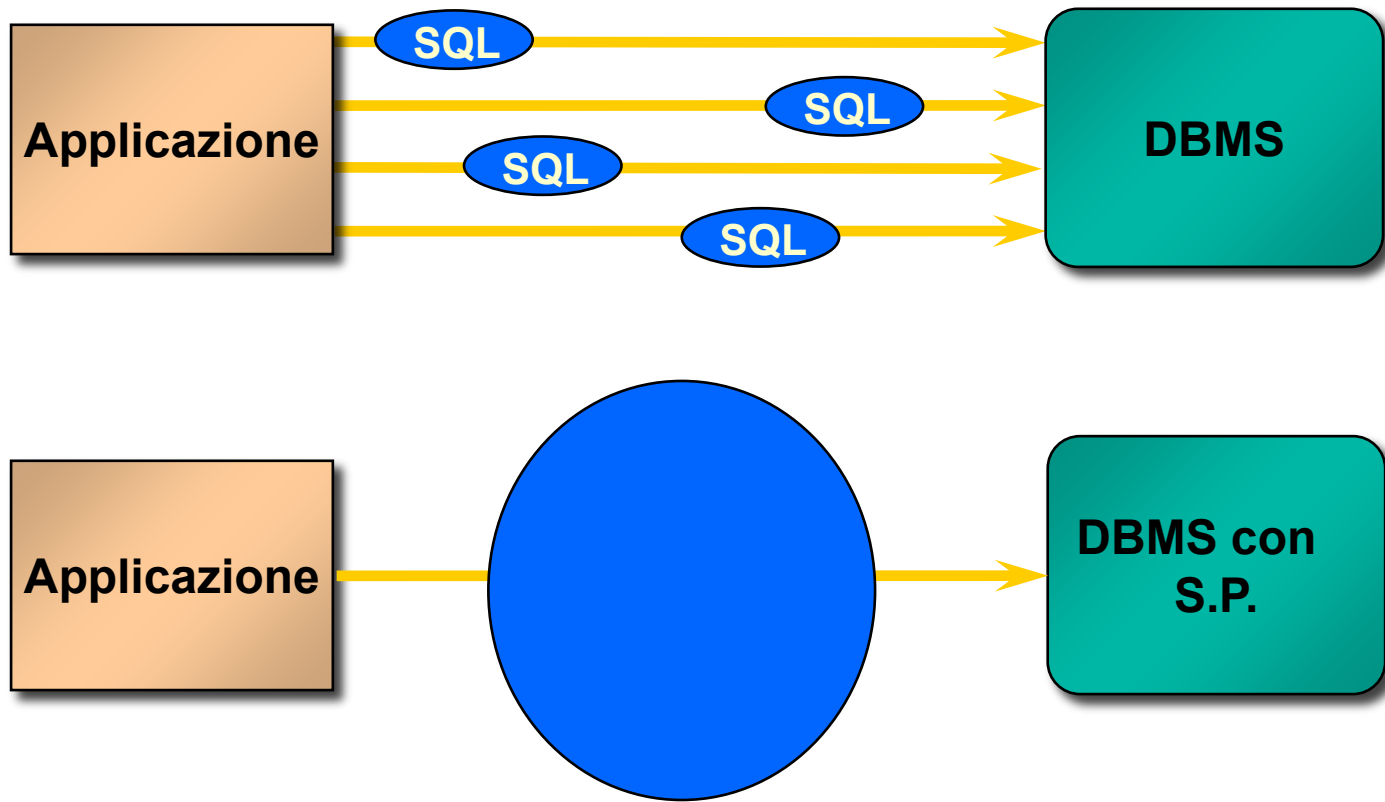
# Vantaggi delle SP

- Le SP sono riusabili e trasparenti a qualsiasi applicazione dato che girano sul server.
- Migliorano l'astrazione (chi invoca la procedura può ignorarne i dettagli implementativi)
- Accesso controllato alle tabelle in quanto solo gli utenti a cui è concesso l'uso della procedura potranno effettuare letture o modifiche alla tabella.
- Controllare centralizzato su certi vincoli d'integrità non esprimibili nelle tabelle.



# Vantaggi delle S.P.

- Prestazioni Elevate



## Svantaggi delle S.P.

- I DBMS utilizzano linguaggi dedicati e differenti per la stesura di procedure.
- 
- La logica dell'applicazione è spostata sul server che dovrà essere in grado di sostenere il carico di lavoro.
- MySQL permette di definire soltanto procedure scritte in linguaggio SQL. Quindi non si ha libertà di scegliere il linguaggio da utilizzare.

# Stored Procedure in MySQL

#definizione di una procedura

**CREATE**

[DEFINER = { user | CURRENT\_USER }]

**PROCEDURE** nome\_procedura ([parametri[,...]])

[caratteristiche ...] corpo\_della\_routine

#definizione UDF

**CREATE** [DEFINER = { user | CURRENT\_USER }]

**FUNCTION** nome\_procedura ([parametri[,...]])

**RETURNS** type

[caratteristiche ...] corpo\_della\_routine

# Stored Procedure in MySQL

**DEFINER** non è un parametro obbligatorio; serve per assegnare un proprietario alla routine.

Nel caso in cui questo non venga specificato verrà considerato come owner predefinito l'utente corrente.

La clausola **RETURNS** è valida per la sintassi delle UDF, queste infatti restituiscono un valore e per questo ad esso è associato il relativo tipo di dato.

# Parametri di una S.P in MySQL

**IN:** rappresenta gli argomenti in ingresso della routine; a questo parametro viene assegnato un valore quando viene invocata la stored procedure; il parametro utilizzato non subirà in seguito modifiche.

- **OUT (non è standard):** è il parametro relativo ai valori che vengono assegnati con l'uscita dalla procedura; questi parametri diventano disponibili per gli utenti per eseguire ulteriori elaborazioni.
- **INOUT (non è standard):** rappresenta una combinazione tra i due parametri precedenti.

# Stored Procedure in MySQL

Quando si definisce una Stored Procedure dalla linea di comando, è necessario introdurre un nuovo delimitatore per terminare il blocco di istruzioni:

```
mysql> delimiter //  
mysql> CREATE PROCEDURE nome_procedura (p1 INT)  
-> BEGIN  
-> blocco istruzioni  
-> END  
-> //  
mysql> delimiter ;
```

# Stored Procedure in MySQL

- In pratica l'istruzione **DELIMITER** iniziale ha lo scopo di comunicare a MySQL che (fino a quando non verrà ordinato diversamente) il delimitatore utilizzato alla fine dell'istruzione non sarà più il “punto e virgola”.
- Per riportare il delimitatore al suo standard, ricordiamoci di dare il comando:

...

```
mysql> delimiter ;
```

# Variabili

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

I tipi delle variabili sono quelli primitivi supportati da MySQL: INT, VARCHAR, DATETIME, ... .

Quando viene dichiarata una variabile, il suo valore iniziale è NULL. E' possibile assegnare un valore utilizzando il comando DEFAULT.

```
DECLARE total_sale INT DEFAULT 0
```

```
DECLARE x, y INT DEFAULT 0
```



# Assegnazione Variabili

```
DECLARE total_count INT DEFAULT 0  
SET total_count = 10;
```

Un altro modo per assegnare un valore ad una variabile è tramite l'istruzione SQL:

```
DECLARE total_products INT DEFAULT 0  
SELECT COUNT(*) INTO total_products  
FROM products
```

# Variabili

- Consigli
  - Seguire una convenzione per i nomi.
  - Inizializzare variabili a NOT NULL.
  - Inizializzare identificatori attraverso l'uso dell'operatore di assegnamento (SET) o con la parola chiave DEFAULT.
  - Dichiarare al più un identificatore per linea.

# Scope delle variabili

Ogni variabile ha il suo proprio scope (visibilità).

Se dichiariamo una variabile dentro una stored procedure, questa sarà visibile (ed utilizzabile) fin quando non si conclude la procedura con il comando END.

E' possibile dichiarare due o più variabili con lo stesso nome in scope differenti; la variabile potrà essere usata all'interno del suo scope.

Una variabile preceduta dal simbolo '@' è una variabile di sessione e persisterà fino alla chiusura di ogni sessione.

# Blocchi annidati e scope delle variabili

## Esempio

```
• ...  
• DECLARE x INT (3) ;  
• BEGIN  
•     ...  
•     DECLARE  
•         y INT ;  
•     BEGIN  
•         ...  
•     END ;  
•     ...  
• END ;
```

Scope of x

Scope of y

# Stored Procedure in MySQL

- “Ciao Mondo” ...

```
mysql> CREATE PROCEDURE proc_ciao () SELECT 'Ciao Mondo' //
```

```
mysql> CALL proc_ciao ()//
```

```
+-----+  
| Ciao Mondo |  
+-----+  
| Ciao Mondo |  
+-----+
```

# Stored Procedure in MySQL

- ... altro esempio

```
DELIMITER //  
CREATE PROCEDURE GetAllProducts()  
    BEGIN  
        SELECT * FROM products;  
    END //  
DELIMITER ;
```

# Stored Procedure in MySQL

- Uso del parametro IN

```
mysql> CREATE PROCEDURE proc_in (IN p INT(3)) SET @x = p//
```

```
mysql> CALL proc_in (14)//
```

```
mysql> SELECT @x//
```

```
+-----+  
| @x    |  
+-----+  
| 14    |  
+-----+
```

# Stored Procedure in MySQL

- Nel corpo della procedura è stata impostata una variabile `x` il cui valore sarà qualsiasi valore associato al parametro `p`.
- Passando come parametro alla procedura l'intero 14 (che viene associato a `p`), sarà possibile ottenere lo stesso valore interrogando il database con una `SELECT` applicata alla variabile `x`.



# Stored Procedure in MySQL

- ... altro esempio

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry
    (IN countryName VARCHAR(255))
    BEGIN
        SELECT city, phone
        FROM offices
        WHERE country = countryName;
    END //
DELIMITER ;
```

```
mysql> CALL GetOfficeByCountry('USA')
```

# Stored Procedure in MySQL

- Uso del parametro OUT

```
mysql> CREATE PROCEDURE proc_out (OUT p INT)
-> SET p = -2 //
```

```
mysql> CALL proc_out (@y) //
```

```
mysql> SELECT @y //
```

```
+-----+
| @y    |
+-----+
| -2    |
+-----+
```

# Stored Procedure in MySQL

- In questo caso p rappresenta il nome di un parametro di output che viene passato come valore alla variabile y introdotta nel momento in cui viene espresso il comando che invoca la procedura.
- Nel corpo della routine il valore del parametro viene indicato come pari all'intero negativo -2, a questo punto spetta ad OUT segnalare a MySQL che il valore sarà associato tramite la procedura.

# Stored Procedure in MySQL

- Uso dei parametri IN e OUT

```
DELIMITER $$  
    CREATE PROCEDURE CountOrderByStatus  
    ( IN orderStatus VARCHAR(25), OUT total INT)  
    BEGIN  
        SELECT count(orderNumber) INTO total  
        FROM orders  
        WHERE status = orderStatus;  
    END$$  
DELIMITER ;
```

```
mysql> CALL CountOrderByStatus('Shipped',@total);  
mysql> SELECT @total AS total_shipped;
```

# Stored Procedure in MySQL

- Uso del parametro INOUT

```

DELIMITER $$
CREATE PROCEDURE `Capitalize` (INOUT str VARCHAR(1024))
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE myc, pc CHAR(1);
    DECLARE outstr VARCHAR(1000) DEFAULT str;
    WHILE i <= CHAR_LENGTH(str) DO
        SET myc = SUBSTRING(str, i, 1);
        SET pc = CASE WHEN i = 1 THEN ' '
        ELSE SUBSTRING(str, i - 1, 1) END;
        IF pc IN (' ', '&', ' ', '_', '?', ';',
            ':', '!', ',', '-', '/', '(', '.') THEN
            SET outstr = INSERT(outstr, i, 1, UPPER(myc));
        END IF;
        SET i = i + 1;
    END WHILE;
    SET str = outstr;
END$$
DELIMITER ;

```

# Stored Procedure in MySQL

- Richiamare la procedura con un parametro di INOUT:

```
mysql> SET @str = 'mysql stored procedure tutorial';  
mysql> CALL Capitalize(@str);  
mysql> SELECT @str;
```

La variabile @str conterrà il valore 'Mysql Stored Procedure Tutorial'

# Controlli Condizionali

I controlli condizionali permettono di eseguire parti di codice secondo il valore di un espressione logica (espressione che utilizza operatori logici e che può essere vera o falsa).

MySQL consente due dichiarazioni condizionali:

- **IF ... THEN ... ELSE ...**
- **CASE WHEN ... THEN ... ELSE ...**

# Controlli Condizionali

```
IF expression THEN commands
    [ELSEIF expression THEN commands]
    [ELSE commands]
END IF;
```

```
IF expression THEN commands
END IF;
```

```
IF expression THEN commands
    ELSEIF expression THEN commands
    ELSE commands
END IF;
```



# Controlli Condizionali

L'utilizzo del comando IF in alcuni casi riduce la leggibilità del codice (ad es. quando vengono scritti IF annidati o di seguito). In questi casi è consigliabile usare il comando **CASE**.

```
CASE
    WHEN expression THEN commands
    ...
    WHEN expression THEN commands
    ELSE commands
END CASE;
```

# Loop

Le stored procedure di MySQL consentono la definizione di loop per consentire di processare comandi iterativamente.

I cicli consentiti sono:

- **WHILE... DO... ELSE ...**
- **REPEAT ... UNTIL ...**

# Loop: WHILE

```
WHILE espressione DO  
    Istruzione  
END WHILE
```

La prima cosa è valutare l'espressione: se è vera viene eseguita l'istruzione fino a che l'espressione non diventa falsa.

# Loop: WHILE

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    WHILE x <= 5 DO
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    END WHILE;
    SELECT str;
END$$
DELIMITER ;
```

# Loop: REPEAT

```
REPEAT  
    Istruzione;  
    UNTIL Espressione  
END REPEAT
```

Inizialmente viene eseguita l'istruzione e poi viene valutata l'espressione. Se l'espressione risulta vera, viene nuovamente rieseguita l'istruzione sino a quando l'espressione non diventa falsa.

# Loop: REPEAT

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    REPEAT
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    UNTIL x > 5
    END REPEAT;
    SELECT str;

END$$
DELIMITER ;
```

# Loop: loop

```
DELIMITER $$
DROP PROCEDURE IF EXISTS LOOPLoopProc$$
CREATE PROCEDURE LOOPLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    loop_label: LOOP
        IF x > 10 THEN
            LEAVE loop_label;
        END IF;
        SET x = x + 1;
        IF (x mod 2) THEN
            ITERATE loop_label;
        ELSE
            SET str = CONCAT(str,x,',');
        END IF;
    END LOOP;
    SELECT str;
END$$
```

# Cursori

MySQL supporta i cursori nelle stored procedures, nelle funzioni e nei triggers.

I cursori vengono usati per setacciare un insieme di righe restituite da una query e permette di controllarle individualmente.



# Cursori

A partire da MySQL 5.x, i cursori hanno le seguenti proprietà:

- **Read only:** non è possibile modificare il contenuto di un cursore.
- **Non-scrollable:** è possibile leggere i dati nel cursore solo in avanti e senza salti in maniera sequenziale.
- **Asensitive:** evitare di aggiornare le tabelle sulle quali sono stati aperti dei cursori: in questi casi è possibile ottenere dalla lettura dei cursori risultati sbagliati.

# Cursori

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

Dopo la dichiarazione di un cursore, esso va aperto con il comando OPEN. Prima di leggerne le righe, occorre aprire il cursore.

```
OPEN cursor_name;
```

Successivamente recuperiamo le righe con il comando FETCH.

```
FETCH cursor_name INTO variable list;
```

# Cursori

Per chiudere e rilasciare la memoria occupata, usare il comando  
CLOSE:

```
CLOSE cursor_name;
```

# Cursori

Il comando NOT FOUND handler serve per evitare che si verifichi la condizione fatale “no data to fetch”:

```
DELIMITER $$  
    DROP PROCEDURE IF EXISTS CursorProc$$  
    CREATE PROCEDURE CursorProc()  
    BEGIN  
        DECLARE no_more_products, quantity_in_stock INT  
            DEFAULT 0;  
        DECLARE prd_code VARCHAR(255);  
        DECLARE cur_product CURSOR FOR  
            SELECT productCode FROM products;  
        DECLARE CONTINUE HANDLER FOR  
            NOT FOUND SET no_more_products = 1;
```

...

# Cursori

```
...  
OPEN cur_product;  
REPEAT  
    FETCH cur_product INTO prd_code;  
    SELECT quantityInStock INTO quantity_in_stock  
    FROM products  
    WHERE productCode = prd_code;  
UNTIL no_more_products = 1  
END REPEAT;  
CLOSE cur_product;
```

...

# MySQL Special Topics

# Obiettivi

- Window Function
- Import dei dati in una tabella MySQL
- Export dei dati in data da una tabella MySQL
- Comprendere problematiche che influenzano le prestazioni
- Analizzare tabelle
- Ottimizzare query

# Window Function

Una window function performa operazioni simili a quelle di aggregazione su un insieme di righe.

A differenza delle funzioni di aggregazione che raggruppano le righe in una sola in base ad un criterio:

- Una window function produce un risultato per ogni riga della query (chiamata riga corrente)
- Le righe su cui esegue l'operazione sono determinate a partire dalla riga corrente usando dei criteri (tali righe si chiamano window)



# Window Function

Esempio 1:

Con raggruppamento

```
mysql> SELECT SUM(profit) AS total_profit  
        FROM sales;
```

```
+-----+  
| total_profit |  
+-----+  
|          7535 |  
+-----+
```

```
mysql> SELECT country, SUM(profit) AS country_profit  
        FROM sales  
        GROUP BY country  
        ORDER BY country;
```

```
+-----+-----+  
| country | country_profit |  
+-----+-----+  
| Finland |          1610 |  
| India   |          1350 |  
| USA     |          4575 |  
+-----+-----+
```

# Window Function

Esempio 1:

Con window function

```
mysql> SELECT
    year, country, product, profit,
    SUM(profit) OVER() AS total_profit,
    SUM(profit) OVER(PARTITION BY country) AS country_profit
FROM sales
ORDER BY country, year, product, profit;
```

year	country	product	profit	total_profit	country_profit
2000	Finland	Computer	1500	7535	1610
2000	Finland	Phone	100	7535	1610
2001	Finland	Phone	10	7535	1610
2000	India	Calculator	75	7535	1350
2000	India	Calculator	75	7535	1350
2000	India	Computer	1200	7535	1350
2000	USA	Calculator	75	7535	4575
2000	USA	Computer	1500	7535	4575
2001	USA	Calculator	50	7535	4575
2001	USA	Computer	1200	7535	4575
2001	USA	Computer	1500	7535	4575
2001	USA	TV	100	7535	4575
2001	USA	TV	150	7535	4575

# Window Function

La finestra è definita dall'inclusione della clausola OVER che specifica come partizionare le righe del risultato per processarle.

Una clausola OVER vuota tratta l'intero insieme di righe come una singola partizione (producendo quindi la somma globale)

Le window function sono permesse SOLO nella target list della SELECT e nella ORDER BY.

Il partizionamento delle righe avviene dopo l'esecuzione di FROM, WHERE, GROUP BY, e HAVING e prima di ORDER BY, LIMIT e DISTINCT.

Le funzioni di aggregazione possono anche essere usate come window function purchè sia presente la clausola OVER.

# Window Function

**CUME\_DIST()**: Distribuzione cumulativa

**DENSE\_RANK()**: Rango della riga corrente all'interno della finestra, senza gap.

**FIRST\_VALUE(expr)**: valore di expr calcolato sulla prima riga della finestra

**LAST\_VALUE(expr)**: valore di expr calcolato sull'ultima riga della finestra

**NTH\_VALUE(expr, n)**: valore di expr calcolato sulla n-esima riga della finestra

**RANK()**: Rango della riga corrente all'interno della finestra, con gap.

**ROW\_NUMBER()**: Numero della riga corrente all'interno della finestra

# Window Function

Esempio:

```
mysql> SELECT
    val,
    ROW_NUMBER() OVER w AS 'row_number',
    RANK() OVER w AS 'rank',
    DENSE_RANK() OVER w AS 'dense_rank'
  FROM numbers
  WINDOW w AS (ORDER BY val);
```

val	row_number	rank	dense_rank
1	1	1	1
1	2	1	1
2	3	3	2
3	4	4	3
3	5	4	3
3	6	4	3
4	7	7	4
4	8	7	4
5	9	9	5

# Definizione della finestra

La finestra si può definire direttamente sulla clausola OVER:

```
SELECT ..., FUNZIONE() OVER(...), ... FROM ...
```

Oppure in una clausola WINDOW separata:

```
SELECT ..., FUNZIONE() OVER nome_finestra, ...  
FROM ...  
WINDOW nome1 AS (...)[, nome2 AS (...), ...]
```

# Definizione della finestra

Come si partizionano le righe nella finestra:

[PARTITION BY field] [ORDER BY field] [frame\_extent]

[frame\_extent] è definito come:

ROWS {frame\_start | BETWEEN frame\_start AND frame\_end}

dove *frame\_start* può essere:

CURRENT\_ROW  
UNBOUNDED PRECEDING  
**N** PRECEDING

*frame\_end* può essere:

CURRENT\_ROW  
UNBOUNDED FOLLOWING  
**N** FOLLOWING

# Definizione della finestra

Esempi:

PARTITION BY subject ORDER BY time  
ROWS UNBOUNDED PRECEDING

PARTITION BY subject ORDER BY time  
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING



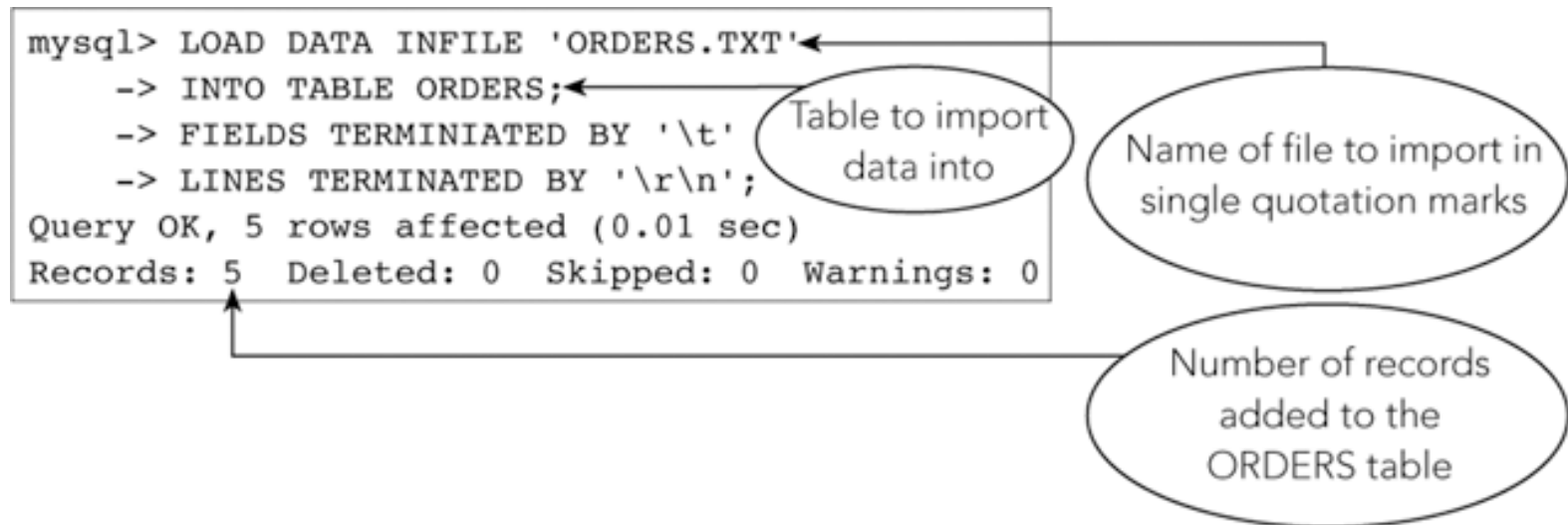
# Window Function

Name	Description
<u>CUME_DIST()</u>	Cumulative distribution value
<u>DENSE_RANK()</u>	Rank of current row within its partition, without gaps
<u>FIRST_VALUE()</u>	Value of argument from first row of window frame
<u>LAG()</u>	Value of argument from row lagging current row within partition
<u>LAST_VALUE()</u>	Value of argument from last row of window frame
<u>LEAD()</u>	Value of argument from row leading current row within partition
<u>NTH_VALUE()</u>	Value of argument from N-th row of window frame
<u>NTILE()</u>	Bucket number of current row within its partition.
<u>PERCENT_RANK()</u>	Percentage rank value
<u>RANK()</u>	Rank of current row within its partition, with gaps
<u>ROW_NUMBER()</u>	Number of current row within its partition

# Group Function

Name	Description
<u>AVG ()</u>	Return the average value of the argument
<u>BIT_AND ()</u>	Return bitwise AND
<u>BIT_OR ()</u>	Return bitwise OR
<u>BIT_XOR ()</u>	Return bitwise XOR
<u>COUNT ()</u>	Return a count of the number of rows returned
<u>COUNT (DISTINCT)</u>	Return the count of a number of different values
<u>GROUP_CONCAT ()</u>	Return a concatenated string
<u>JSON_ARRAYAGG ()</u>	Return result set as a single JSON array
<u>JSON_OBJECTAGG ()</u>	Return result set as a single JSON object
<u>MAX ()</u>	Return the maximum value
<u>MIN ()</u>	Return the minimum value
<u>STD ()</u>	Return the population standard deviation
<u>STDDEV ()</u>	Return the population standard deviation
<u>STDDEV_POP ()</u>	Return the population standard deviation
<u>STDDEV_SAMP ()</u>	Return the sample standard deviation
<u>SUM ()</u>	Return the sum
<u>VAR_POP ()</u>	Return the population standard variance
<u>VAR_SAMP ()</u>	Return the sample variance
<u>VARIANCE ()</u>	Return the population standard variance

# Importing Data into a Database




**FIGURE 8-2** Importing a text file into the ORDERS table

# Importing Data into a Database (continued)

```
mysql> LOAD DATA INFILE 'WARE.TXT'
-> INTO TABLE WAREHOUSE
-> FIELDS TERMINATED BY '\t'
-> LINES TERMINATED BY '\r\n';
Query OK, 3 rows affected (0.00 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
mysql> SELECT *
-> FROM WAREHOUSE;
```

WAREHOUSE	LOCATION
1	East
2	West
3	North

3 rows in set (0.00 sec)



**FIGURE 8-4** Importing text data into the WAREHOUSE table

# LOAD DATA

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'  
  [REPLACE | IGNORE]  
  INTO TABLE tbl_name  
  [FIELDS  
    [TERMINATED BY '\t']  
    [[OPTIONALLY] ENCLOSED BY '']  
    [ESCAPED BY '\\\']  
  ]  
  [LINES  
    [STARTING BY '']  
    [TERMINATED BY '\n']  
  ]  
  [IGNORE number LINES]  
  [(col_name,...)]
```

- Il comando LOAD DATA INFILE consente il caricamento di una tabella ad alta velocità leggendo le righe da un file di testo.

- Caricamento dei dati:

```
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO  
TABLE pet;
```

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO  
TABLE event;
```

```
mysql> LOAD DATA INFILE '/tmp/test.txt'  
      -> INTO TABLE test LINES STARTING BY  
      "yyy";
```

- Quindi un file contenente

```
yyy"Row",1
```

```
blablabla yyy"Row",2
```

- Può essere letto e verrà caricato come ("row",1), ("row",2)

# EXPLAIN

```
EXPLAIN tbl_name
```

Oppure

```
EXPLAIN SELECT select_options
```



# EXPLAIN

- Consente di capire le performance di una query e mostra quali indici effettivamente la query sta usando.

# Le colonne indicizzate devono essere stand alone

```
mysql> explain select * from event where year(event.date) < '2003';
```

table	type	possible_keys	key	key_len	ref	rows	Extra
event	ALL	NULL	NULL	NULL	NULL	6	Using where

1 row in set (0.00 sec)

```
mysql> explain select * from event where event.date < '2003-01-01';
```

table	type	possible_keys	key	key_len	ref	rows	Extra
event	ALL	Index_2	NULL	NULL	NULL	6	Using where

1 row in set (0.00 sec)

# Esercizio

Selezionare un dataset a piacere da questi siti:

<https://dati.regione.sicilia.it/>

[Home Page | dati.gov.it](#)

Creare un database per ospitare questi dati e usare il loader per importarli nel database.