Progettazione Fisica e SQL

Ing. Alessandro Pellegrini, PhD
pellegrini@diag.uniroma1.it

Creazione Stored Procedure

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement
```

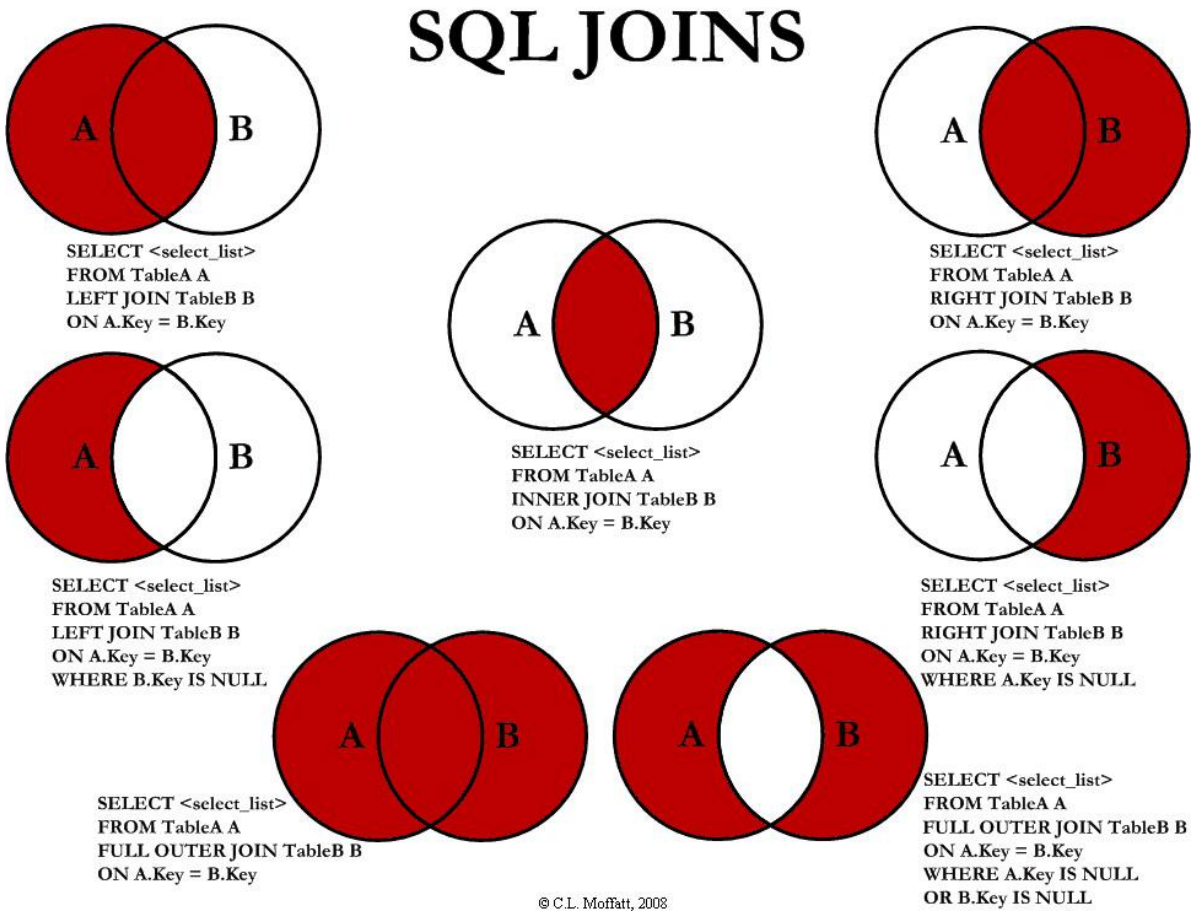
Indici

```
CREATE [OR REPLACE] [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Select

```
SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [ FROM table_references
      [WHERE where_condition]
      [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_condition]
      [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [INTO OUTFILE 'file_name' [CHARACTER SET charset_name] [export_options]
```

Le operazioni di Join



Livelli di Isolamento

- **READ UNCOMMITTED**
- Gli statement SELECT sono eseguiti in modalità non bloccante, ma è possibile che venga utilizzata una versione precedente di una riga. Pertanto, utilizzando questo livello di isolamento, le letture possono non esere coerenti (dirty read).
- Questo fenomeno si verifica poiché una transazione può leggere dati da una riga aggiornata da un'altra transazione che non ha ancora eseguito l'operazione di commit.

Dirty Reads

T1	T2
SELECT age FROM users WHERE id = 1;	
	UPDATE users SET age = 21 WHERE id = 1;
SELECT age FROM users WHERE id = 1;	
	ROLLBACK;

Livelli di Isolamento

- **READ COMMITTED**
- Il DBMS acqusisce un lock per ogni dato che viene letto o scritto. I lock associati ai dati che sono stati aggiornati in scrittura sono mantenuti fino alla fine della transazione, mentre i lock associati ai dati acceduti in lettura sono rilasciati alla fine della singola lettura.

- Possono verificarsi anomalie di tipo *unrepeatable reads*.

Unrepeatable Reads

T1	T2
SELECT * FROM users WHERE id = 1;	
	UPDATE users SET age = 21 WHERE id = 1;
	COMMIT;
SELECT * FROM users WHERE id = 1;	
COMMIT;	

Livelli di Isolamento

- **REPEATABLE READ**
- Con questo livello di isolamento, vengono mantenuti i lock sia dei dati acceduti in lettura sia in scrittura fino alla fine della transazione. Non vengono però gestiti i *range lock*, pertanto possono verificarsi anomalie di tipo *phantom read*.

- Le *phantom read* sono associate ad inserimenti che avvengono in concorrenza.

- Nelle versioni più nuove di InnoDB, è il livello di isolamento predefinito.

Phantom Reads

T1	T2
SELECT * FROM users WHERE age BETWEEN 10 AND 30;	
	INSERT INTO users(id,name,age) VALUES (3, 'Bob', 27);
	COMMIT;
SELECT * FROM users WHERE age BETWEEN 10 AND 30;	
COMMIT;	

Livelli di Isolamento

- **SERIALIZABLE**
- Tutti i lock vengono mantenuti fino alla fine della transazione e ogni volta che una SELECT utilizza uno specificatore di tipo WHERE, viene acquisito anche il *range lock*.

- In sistemi non basati su lock, questo livello di isolamento può essere implementato mediante il concetto di *read/write set* o in generale di *multiversion concurrency control*.

Quale scegliere?

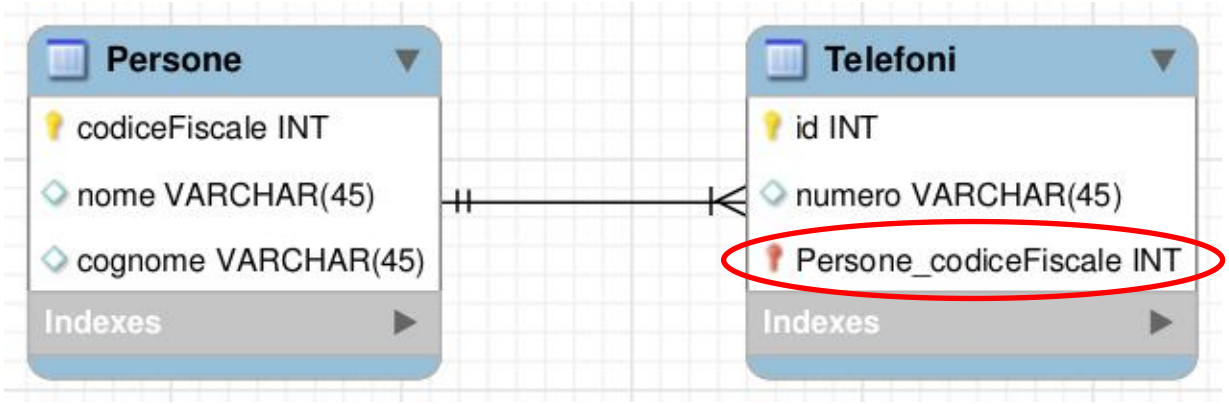
1. Per molte applicazioni, la maggior parte delle transazioni possono essere costruite in maniera tale da non richiedere l'utilizzo di livelli di isolamento molto alti (ad esempio SERIALIZABLE), riducendo l'overhead dovuto ai lock
2. Lo sviluppatore deve accertarsi con cautela che le modalità di accesso delle transazioni non causino bug software dovuti alla concorrenza e al rilassamento dei livelli di isolamento.
3. Se si utilizzano unicamente alti livelli di isolamento, la probabilità di *deadlock* cresce notevolvmente.

Stored Procedure Types

Type Value	Type Description
MYSQL_TYPE_TINY	TINYINT field
MYSQL_TYPE_SHORT	SMALLINT field
MYSQL_TYPE_LONG	INTEGER field
MYSQL_TYPE_INT24	MEDIUMINT field
MYSQL_TYPE_LONGLONG	BIGINT field
MYSQL_TYPE_DECIMAL	DECIMAL or NUMERIC field
MYSQL_TYPE_NEWDECIMAL	Precision math DECIMAL or NUMERIC
MYSQL_TYPE_FLOAT	FLOAT field
MYSQL_TYPE_DOUBLE	DOUBLE or REAL field
MYSQL_TYPE_BIT	BIT field
MYSQL_TYPE_TIMESTAMP	TIMESTAMP field
MYSQL_TYPE_DATE	DATE field
MYSQL_TYPE_TIME	TIME field
MYSQL_TYPE_DATETIME	DATETIME field
MYSQL_TYPE_YEAR	YEAR field
MYSQL_TYPE_STRING	CHAR or BINARY field
MYSQL_TYPE_VAR_STRING	VARCHAR or VARBINARY field
MYSQL_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
MYSQL_TYPE_SET	SET field
MYSQL_TYPE_ENUM	ENUM field
MYSQL_TYPE_GEOMETRY	Spatial field
MYSQL_TYPE_NULL	NULL-type field

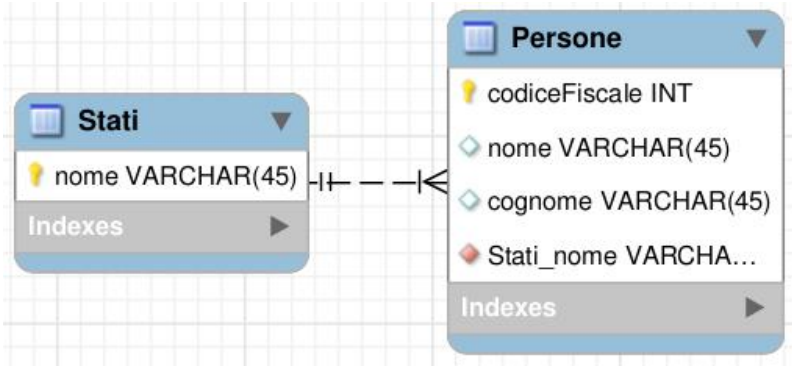
Identifying / Non-Identifying Relationships

- Una *Identifying Relationship* identifica quando l'esistenza di una riga in una tabella “figlia” dipende da una riga in una tabella “padre”.
- La relazione logica dice che un figlio non può esistere senza il padre

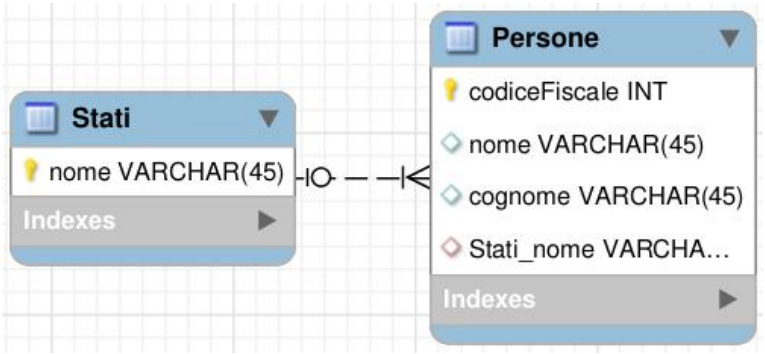


Identifying / Non-Identifying Relationships

- Una *Non-Identifying Relationship* identifica relazioni tra righe di tabelle che hanno esistono logicamente indipendentemente l'una dall'altra.



Obbligatoria



Opzionale

Utilizzo di Trigger per emulare le Asserzioni

- MySQL non supporta le asserzioni (in particolare il comando `STOP ACTION` non è supportato)
- Si possono utilizzare trigger “before update” per emulare il funzionamento delle asserzioni

```
create assertion AT_MOST_ONE_MANAGER as CHECK
((select count(*)  from `employees` E
   where E.ruolo = 'MANAGER') <= 1
)
```

Utilizzo di Trigger per emulare le Asserzioni

```
create trigger AT_MOST_ONE_MAGANGER
before insert on `employees` for each row
begin
	declare counter INT;
	select count(*) from `employees` E
	where E.ruolo = 'MANAGER' into counter;
	if counter > 1 then
		signal sqlstate '45000';
	end if;
end
```