Database Management Systems:
Fundamentals and Introduction to SQL

Advanced Topics

MySQL-Based Programming

- MySQL client API
- Connecting/Disconnecting to the MySQL server
- Executing SQL statements
- Retrieving results

Language	Interface
Perl	Perl DBI
Ruby	Mysql2 gem
PHP	PDO
Python	DB-API
Go	Go sql
Java	JDBC

- https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html

Stored Procedure

- A stored function or procedure object encapsulates the code for performing an operation, enabling you to invoke the object easily by name rather than repeat all its code each time it's needed.
  - store and organize SQL
  - faster execution, SQL optimization
  - data security, e.g. remove direct data access

Stored Procedure

```
delimiter $$
# define a new delimiter to define the end of the entire procedure, but inside it, individual statements are each terminated by ;
CREATE PROCEDURE proc_name()
        BEGIN
                ...
                ...;

        END$$

delimiter ;

CALL proc_name();
```

Main operation you want to store in the procedure

Trigger

- *A trigger is an object that activates when a table is modified by an INSERT, UPDATE, or DELETE statement*

- check values before they are inserted into a table
- specify that any row deleted from a table should be logged to another table that serves as a journal of data changes

- A trigger can be set to activate either before or after the trigger event.

Trigger Syntax

- CREATE TRIGGER trigger_name()

```
AFTER  insert | update | delete ON table1
FOR each row
BEGIN
        update table2
        set …
        where… ;
        change actions log
END
```

Event

- An event is an object that executes SQL statements at a scheduled time or times.
- SET GLOBAL event_scheduler = 'ON';

```
CREATE EVENT yearly_delete_stale_payment_logs
ON SCHEDULE
        -- AT '2020-01-01'
        EVERY 1 YEAR -- STARTS '2020-01-01' ENDS '2029-01-01'
DO BEGIN
        DELETE FROM payment_logs
        WHERE action_date < NOW() - INTERVAL 1 YEAR;
END
```

Concurrency Control

- MySQL server can handle multiple clients at the same time because it is multi-threaded
- multiple statements executed are dependent on one another, other clients may be updating tables in between those statements may cause difficulties or problems

Concurrency Control

- Concurrency Control deals with interleaved execution of more than one transaction.
- **Transaction**: a set of logically related operations.
- **Commit**: After all instructions of a transaction are successfully executed, the changes made by transaction are made permanent in the database.
- **Rollback**: If a transaction is not able to execute all operations successfully, all the changes made by transaction are undone.

Properties of Transaction - ACID

- **Atomicity**: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

- *For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.*

Properties of Transaction - ACID

- **Consistency**: Data is in a consistent state when a transaction starts and when it ends.

- *For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.*

Properties of Transaction - ACID

- **Isolation**: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

- *For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.*

Properties of Transaction - ACID

- **Durability**: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

- *For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.*

Transaction Isolation Levels

- **Read Uncommitted**
  - Lowest isolation level; transactions are not isolated from each other.
  - One transaction may read not yet committed changes made by other transaction, thereby allowing **dirty reads**.
- **Read Committed**
  - guarantees that any data read is committed at the moment it is read
  - dirty read not allowed
  - The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.

Dirty Reads

- Dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

Transaction 1	Transaction 2
Update record X	
	Read record X
Rollback	

In the example, if transaction 1 fails for some reason, then X will revert back to its previous value. But transaction 2 has already read the incorrect value of X.

Transaction Isolation Levels (cont)

- **Repeatable Read** (MySQL default): The transaction holds read locks on all rows it references and writes locks on all rows it accesses. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.
- **Serializable**: This is the Highest isolation level. All the rows are locked for the duration of the transaction, no insert, update, or delete is allowed.