

Taller de proyecto 2-TP1

Fecha de entrega: 30/08/2018

—

Integrantes: Aguilar Sergio 765/7
Ailigo Oriana 1202/8

Detalle de todo el proceso de adquisición de datos, ejecución de ambos procesos, esquema general y la interacción del usuario.

Para el proceso de adquisición primero se implementa un proceso (process.py) que posee el método main (samples, session) el cual se encarga de generar los datos de forma aleatoria , estos datos son almacenados en la base de datos local en intervalos de 1 segundo.

NOTA: los datos generados para temperatura,humedad,presión,velocidad del viento varían en un rango de +- 2 (ver figura 1)

```
def main (samples, session):
    s = samples
    temp = random.randint(0, 20)           #genero una sola vez v
    humi = random.randint(60, 80)          #genero una sola vez v
    press = random.randint(1017, 1030)      #genero una sola vez v
    wspeed = random.randint(15, 30)         #genero una sola vez v
    while(True):
        s.temperature = random.randint((temp-2), (temp+2))
        s.humidity = random.randint((humi-2), (humi+2))
        s.pressure = random.randint((press-2), (press+2))
        s.windspeed = random.randint((wspeed-5), (wspeed+5))
        session.add(s)
        session.commit()
        time.sleep(1)
    session.close()
```

figura 1: Función main(samples, session) que genera y almacena valores random.

Por otro lado tenemos un proceso (app.py) que se encarga de arrancar el proceso generador de de datos y de adquirir los datos para presentarlos mediante la página (index.html).

app.py configura la ruta que le da al main e invoca métodos que realizan las consultas a la base de datos mediante un query como se hace en el método get_last(self). Al query se le pasa como argumento la tabla donde se alojan las muestras. Por otro lado, se utiliza un método de ordenamiento para obtener las últimas muestras tal como lo exigen los requerimientos del enunciado. (Ver figura 2)

```
def get_last(self):  
    session = self.get_session()  
    ultima = session.query(Samples).order_by(Samples.id.desc()).first()  
    session.close()  
    return ultima
```

Figura 2: Función `get_last(self)` que obtiene los últimos valores de la tabla de la base de datos.

Otro método que implementa es `get_last_ten(self)` que se almacena en una variable los últimos diez valores de la tabla, donde luego acumula para después realizar el promedio de las misma (**ver figura 3**). Por otro lado, esta el método `get_session(self)` que primero llama al servidor local los cuales recibe los parámetros de usuario, confirma la coneccion, inicia el motor de la base de datos, genera nuevos objetos de session cuando se los llama, creandolos dados los argumentos de configuración establecidos. El objeto Session le permite persistir ciertos parámetros en las solicitudes y por último La clase base contiene un objeto Metadata donde se recopilan los objetos de tabla recientemente definidos. Se pretende acceder directamente a este objeto para operaciones específicas de Metadata, `create_all` crea restricciones de clave externa entre tablas generalmente en línea con la definición de la tabla en si, y por esta razon tambien genera las tablas en orden de su dependencia. (**ver figura 4**)

```
def get_last_ten(self):  
    promedio = Samples()  
    promedio.temperature = 0  
    promedio.humidity = 0  
    promedio.pressure = 0  
    promedio.windspeed = 0  
    session = self.get_session()  
    diez = session.query(Samples).order_by(Samples.id.desc()).limit(10)  
    session.close()  
  
    for d in diez:  
        promedio.temperature = (promedio.temperature + d.temperature)  
        promedio.humidity = (promedio.humidity + d.humidity)  
        promedio.pressure = (promedio.pressure + d.pressure)  
        promedio.windspeed = (promedio.windspeed + d.windspeed)  
    promedio.temperature = (promedio.temperature/10)  
    promedio.humidity = (promedio.humidity/10)  
    promedio.pressure = (promedio.pressure/10)  
    promedio.windspeed = (promedio.windspeed/10)  
    return promedio
```

Figura 3: `get_last_ten(self)` que obtiene el promedio de las últimas 10 muestras.

```

def get_session(self):
    """Singleton of db connection

    Returns:
        [db connection] -- [Singleton of db connection]
    """
    if self.session == None:
        connection = 'mysql+mysqlconnector://%s:%s@%s:%s/%s' % (self.db_user, self.db_pass, self.db_host, self.db_port, self.db_name)
        engine = create_engine(connection, echo=True)
        connection = engine.connect()
        Session = sessionmaker(bind=engine)
        self.session = Session()
        self.Base.metadata.create_all(engine)
    return self.session

```

Figura 4: `get_session(self)` devuelve la session para realizar la conexión con la base de datos.

Interacción del usuario:

La página imprime muestras de velocidad del viento, temperatura, presión y humedad de la plata. Donde se puede observar en la parte izquierda, las muestras y en la parte derecha, los 10 últimos promedios. Cuando el usuario presiona el botón “seleccionar” después de haber introducido algunos de los valores dentro de este rango (1, 2, 5, 10, 30 y 60 segundos), la página se refrescará con nuevos valores. ya que este parámetro introducido se comunica con un formulario que lo envía a una función que genera el refrescar de la página.

Problemática de la concurrencia de la simulación

La problemática de concurrencia existente se establece cuando el usuario desea colocar otro intervalo de tiempo, precisamente de un segundo. El tiempo que lleva en realizar la transacción de dicha petición y la de la generación de muestras es igual. Por lo tanto se genera un conflicto porque se van acumulando peticiones sobre el mismo dato que se va a depositar en la base de datos. En este momento el que deposita el dato tiene prioridad por el que desea obtener el dato. Esto provoca que los pedidos queden en espera acumulandose, produciendo un fallo de transacción.

Una base de datos está en un estado consistente si obedece todas las restricciones de integridad (significa que cuando un registro en una tabla haga referencia a un registro en otra tabla, el registro correspondientes debe existir) definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones y eliminación de información. Por supuesto, se quiere asegurar que la base de datos nunca entre en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente.

Es por ello que se debe asegurar que la base de datos regrese a un estado consistente al fin de la ejecución de una transacción(commit). Además de realizar un control para que la ejecución de transacciones que se operan en paralelo, y acceden a información compartida no interfieran potencialmente unas con otras.

Lo que sucede en el ambiente real.

Ejemplos que se pueden presentar: El hecho de reservar un asiento en una avión mediante un sistema basado en aplicaciones web, cuando decenas de personas en el mundo pueden reservarlo también, nos da una idea de lo importante y crucial que es el control de concurrencia en un sistema de base de datos a mediana o gran escala.

Otro ejemplo en el que podemos observar la incidencia del control de concurrencia en el siguiente: en una Base de Datos bancaria podría ocurrir que se paguen dos cheques en forma simultánea sobre una cuenta que no tiene saldo suficiente para cubrirlos en su totalidad, esto es posible evitarlo si se tiene un control de concurrencia.

¿Podría producirse problemas de concurrencia muy difíciles o imposibles de simular?Comente brevemente los posibles problemas de tiempo real que podrían producirse en general.


Si se podría generar problemas difíciles de simular. Ya que puede considerarse el caso en que haya más datos que funciones para procesar los mismos. Otro caso sería el de escribir y leer sobre el mismo dato. Cuando al realizarse una escritura se tarda mucho en realizarse y hay procesos que necesite de ese dato pero como que no se encuentra disponible se quedan esperando. Por lo tanto, si hay más procesos que esperan por el mismo dato generaría deadlock.

Los posibles problemas de tiempo real son la comunicación entre tareas que se ejecutan en paralelo. Por otro lado, está la determinación de cuándo y cómo intercalar esas tareas que pueden interactuar entre sí (planificación de ejecución de cada tarea).

Por ejemplo, El sistema ABS (anti-lock breaking system) de un auto y un marcapasos.

En un ambiente real no se posee un software que genera datos, sino que se tiene un dispositivo que capta los datos mediante la interacción con el usuario y estos son analizados por software almacenandose en la base de datos.

Los sistemas que tratan el problema de control de concurrencia permiten que sus usuarios asuman que cada una de sus aplicaciones se ejecutan atómicamente, como si no existieran otras aplicaciones ejecutándose concurrentemente. Esta abstracción de una ejecución atómica y confiable de una aplicación se conoce como una **transacción**. Las transacciones se ejecuten



atómicamente controlando la intercalación de transacciones concurrentes, para dar la ilusión de que las transacciones se ejecutan serialmente, una después de la otra, sin ninguna intercalación.

El objetivo del control de concurrencia y recuperación es asegurar que dichas transacciones se ejecuten atómicamente, es decir: Cada transacción accede a información compartida sin interferir con otras transacciones, y si una transacción termina normalmente, todos sus efectos son permanentes, en caso contrario no tiene efecto alguno.

Diferencias entre el funcionamiento del simulador y el sistema real que se está tratando de simular:

Simulación planteada

- Datos recogidos de manera periódica y controlada en tiempo real.
- intenta generar muestras de escenarios imposibles.
- La exactitud con que se predicen datos históricos.
- La exactitud en la predicción del futuro.
- La comprobación de falla del modelo de simulación al utilizar datos que hacen fallar al sistema real.
- se puede variar la velocidad de respuesta con la que cambia el entorno.

Sistemas real

- Latencia de microcontrolador con el servidor: debido a que en la simulación no se puede implementar.
- La recolección de datos no es sincrónica , no posee valores acotados y controlados .
- Costoso, Complejo, y peligroso,