Qual das opções são verdadeiras se a tabela estiver vazia antes de este código ser executado? (Escolha todas as que se aplicam.)

```
var sql = "INSERT INTO people VALUES(?, ?, ?)";
conn.setAutoCommit(false);
try (var ps = conn.prepareStatement(sql,
ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE)) {
    ps.setInt(1, 1);
    ps.setString(2, "Joslyn");
    ps.setString(3, "NY");
    ps.executeUpdate();
    Savepoint sp = conn.setSavepoint();
    ps.setInt(1, 2);
    ps.setString(2, "Kara");
    ps.executeUpdate();
    conn._________________;
}
```

A. Se a linha em branco contiver rollback(), não há linhas na tabela.
B. Se a linha em branco contiver rollback(), há uma linha na tabela.
C. Se a linha em branco contiver rollback(sp), não há linhas na tabela.
D. Se a linha em branco contiver rollback(sp), há uma linha na tabela.
E. O código não compila.
F. O código lança uma exceção porque a segunda atualização não define todos os parâmetros.

BOYARSKY, Jeanne; SELIKOFF, Scott. OCP **Oracle® Certified Professional Java SE 17 Developer Study Guide**. John Wiley & Sons, 2022.

ANÁLISE DA QUESTÃO

O código realiza operações *SQL* e controla a transação com *setAutoCommit(false),* o que significa que nenhuma alteração é confirmada automaticamente; a confirmação deve ser feita manualmente com um *commit()*. O documento da Oracle JDBC Basics sobre o desabilitar o auto commit diz o seguinte:

> "When a connection is created, it is in auto-commit mode. This means **that each individual SQL statement is treated as a transaction and is automatically committed right after it is executed**. (To be more precise, the default is for a SQL statement to be committed when it is completed, not when it is executed. **A statement is completed when all of its result sets and update counts have been retrieved**. In almost all cases, however, a statement is completed, and therefore committed, right after it is executed.)
>
> The way **to allow two or more statements to be grouped into a transaction is to disable the auto-commit mode**. This is demonstrated in the following code, where con is an active connection:
>
> con.setAutoCommit(false);"

1. **Criação da String SQL:**

```
var sql = "INSERT INTO people VALUES(?, ?, ?)";
```

Esse SQL é uma instrução de inserção com três valores parametrizados.

2. **Desativação do Auto Commit:**

```
conn.setAutoCommit(false);
```

Conforem dito anteriormente, indica que todas as operações seguintes serão parte de uma transação que precisará ser confirmada explicitamente.

3. **Preparação da Instrução SQL:**

```
try (var ps = conn.prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE)) {
```

Aqui, preparamos a instrução *SQL* para inserção com parâmetros e especificamos que o *ResultSet* será sensível a alterações (TYPE_SCROLL_SENSITIVE) e que ele permitirá atualizações (CONCUR_UPDATABLE).

4. **Primeira Inserção:**

```
ps.setInt(1, 1);
ps.setString(2, "Joslyn");
ps.setString(3, "NY");
ps.executeUpdate();
```

O código está inserindo um novo registro na tabela *people* com os valores (1, "Joslyn", "NY")

5. **Criação do Savepoint:**

```
Savepoint sp = conn.setSavepoint();
```

Aqui, criamos um ponto de salvamento (sp) na transação. Se precisarmos fazer um rollback para este ponto, podemos voltar a esta situação específica.

6. **Segunda Inserção:**

```
ps.setInt(1, 2);
ps.setString(2, "Kara");
ps.executeUpdate();
```

Tentamos inserir outra linha, mas falta o 3o parâmetro.

Impacto das Operações de Rollback

- ***Rollback sem Savepoint (rollback()):***

Se a linha em branco contiver *rollback(),* toda a transação será revertida, incluindo todas as inserções feitas. Isso resulta na tabela sem nenhuma linha.

A. Rollback para o Savepoint (rollback(sp)):

Se a linha em branco contiver *rollback(sp),* a transação será revertida para o **ponto onde estava após a primeira inserção**. Isso significa que a linha com os valores (1, "Joslyn", "NY") permanecerá na tabela, mas a linha com os valores (2, "Kara", "NY") será descartada.

Agora analisando as alternativas:

A. Se a linha em branco contém rollback(), não há linhas na tabela.

Correto. O rollback sem o savepoint reverte toda a transação, deixando a tabela vazia.Isto elimina portanto a opção B

D. Se a linha em branco contém rollback(sp), há uma linha na tabela.

Correto. O rollback para o savepoint reverte a transação para o ponto após a primeira inserção, preservando a linha com (1, "Joslyn", "NY") e descartando a linha com (2, "Kara", "NY"). Is elimina portanto a opção C.

E. O código não compila.

Incorreto. O código compila corretamente, já que todos os elementos necessários estão presentes.

F. O código lança uma exceção porque a segunda atualização não define todos os parâmetros.

Incorreto. **O JDBC reutiliza os parâmetros definidos anteriormente se não forem sobrescritos**. Portanto, a linha com (2, "Kara", "NY") é válida.

Com isso, as opções corretas são **A** e **D**.

Vamos detalhar um pouco sobre o método *prepareStatement*:

O método *prepareStatement* da interface *Connection* no Java é usado para criar um objeto *PreparedStatement*, que **permite a execução de instruções SQL pré-compiladas com parâmetros**. Isso **melhora o desempenho e a segurança**, pois evita a necessidade de compilar a instrução SQL a cada execução e reduz a vulnerabilidade a ataques de injeção de SQL.

Sintaxe Básica:

```
PreparedStatement prepareStatement(String sql)
```

Essa forma básica cria uma PreparedStatement para a execução da instrução SQL fornecida.

Opções Avançadas com Parâmetros

O método *prepareStatement* também pode ser chamado **com opções avançadas que especificam o comportamento do *ResultSet*** e outras características da instrução SQL. No código fornecido, foram usadas as constantes *ResultSet*.TYPE_SCROLL_SENSITIVE e *ResultSet*.CONCUR_UPDATABLE.

Constantes para Tipo de *ResultSet*

- ***ResultSet*.TYPE_FORWARD_ONLY**: O cursor do *ResultSet* só pode se mover para frente usando o método de movimentação do cursor *next()*. É o padrão e geralmente mais eficiente.
- ***ResultSet*.TYPE_SCROLL_INSENSITIVE**: O cursor pode se mover para frente e para trás, utilizando métodos de movimentação de cursor, mas o *ResultSet* não é sensível a mudanças feitas no banco de dados após sua criação.
- ***ResultSet*.TYPE_SCROLL_SENSITIVE:** O cursor pode se mover para frente e para trás utilizando métodos de movimentação de cursor, e o *ResultSet* é sensível a mudanças no banco de dados que acontecem após sua criação.

Constantes para Concurrency do *ResultSet*

- ***ResultSet*.CONCUR_READ_ONLY:** O *ResultSet* não permite atualizações nos dados. É o padrão.
- ***ResultSet*.CONCUR_UPDATABLE:** O *ResultSet* permite atualizações nos dados, permitindo que você altere o banco de dados diretamente através dele.

Outras Opções e Parâmetros

Além das constantes usadas no exemplo, há outras sobrecargas do método *prepareStatement*:

- ***prepareStatement*(String sql, int autoGeneratedKeys):** Permite especificar se as chaves geradas automaticamente devem ser retornadas (*Statement*.RETURN_GENERATED_KEYS) ou não (*Statement*.NO_GENERATED_KEYS).

- ***prepareStatement*(String sql, int[] columnIndexes):** Retorna as chaves geradas automaticamente para as colunas especificadas.

- ***prepareStatement*(String sql, String[] columnNames):** Retorna as chaves geradas automaticamente para as colunas com os nomes especificados.

- ***prepareStatement*(String sql, int *resultSetType*, int *resultSetConcurrency*, int *resultSetHoldability*):** Especifica o comportamento do ResultSet em relação ao tipo, concorrência e a capacidade de retenção (*ResultSet*.CLOSE_CURSORS_AT_COMMIT ou *ResultSet*.HOLD_CURSORS_OVER_COMMIT).

Vamos ver um examplo com a geração de chaves, o argumento *Statement*.RETURN_GENERATED_KEYS é uma constante da interface *Statement* que **indica que as chaves geradas automaticamente pelo banco de dados devem ser retornadas**. Isso é útil, por exemplo, se a **tabela people tiver uma coluna de chave primária que seja preenchida automaticamente pelo banco de dados.**

Exemplo Prático

Vamos ver um exemplo completo de como esse código pode ser usado usando *Statement*.RETURN_GENERATED_KEYS*:*

```
 import java.sql.*;

public class InsertPerson {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "username";
        String password = "password";

        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            String sql = "INSERT INTO people (name, location) VALUES(?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql,
                   Statement.RETURN_GENERATED_KEYS);

            ps.setString(1, "John Doe");
            ps.setString(2, "New York");

            int affectedRows = ps.executeUpdate();
            if (affectedRows > 0) {
                // Obtém a(s) chave(s) gerada(s) automaticamente
                try (ResultSet generatedKeys = ps.getGeneratedKeys()) {
                    while (generatedKeys.next()) // move o cursor sobre o conjunto de
                                                 // chaves geradas
                        {
                        long id = generatedKeys.getLong(1);
                        System.out.println("Registro inserido com ID: " + id);
                    }
                }
            } else {
                System.out.println("Nenhuma linha afetada.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

Explicando passo a passo o código:

1. **Configuração da Conexão:**

   A URL, o nome de usuário e a senha são usados para conectar ao banco de dados.

2. **Preparação da Instrução:**

   A instrução SQL é preparada com *placeholders* (?) que serão substituídos pelos valores fornecidos pelo usuário.

3. **Substituição dos *Placeholders*:**

   Os métodos *ps.setString*(1, "John Doe") e *ps.setString*(2, "New York") substituem os ? na instrução SQL pelos valores "John Doe" e "New York".

4. **Execução da Instrução:**

   O método *ps.executeUpdate()* executa a instrução de inserção. Ele retorna o número de linhas afetadas pela instrução.

5. **Recuperação da Chave Gerada:**

   O *ps.getGeneratedKeys()* retorna um *ResultSet* contendo as chaves geradas. O código verifica se há uma ou mais chaves disponíveis movimentando o cursor via *next()* e as imprime.

Veja que não é estritamente necessário envolver a chamada para *ps.getGeneratedKeys()* em um bloco *try-catch*, pois, geralmente, esse método não lança exceções. No entanto, o uso de *try-catch* é uma **boa prática para capturar exceções inesperadas** que possam ocorrer durante operações de banco de dados, garantindo que o código se comporte de maneira controlada em caso de falhas.

Perceba também que não há necessidade de um catch separado para o *try* interno que lida com o *ResultSet* (*generatedKeys*), porque ele está dentro do *try* externo, que já captura todas as exceções *SQLException*. O *try* interno com *ResultSet* é usado principalmente para garantir que o recurso seja fechado automaticamente, o que é uma prática recomendada para evitar vazamentos de recursos.

Existe um gerenciamento automático de recursos feito pelo bloco *try* (*try-with-resources*) em Java no contexto de JDBC. O *try-with-resources* foi introduzido no Java 7 e simplifica o gerenciamento de recursos que implementam as interfaces *AutoCloseable* ou *Closeable*. **Isso permite que recursos sejam fechados de forma automática e segura ao final do bloco try.**

ResultSet e *Connection* são subclasess de *AutoCloseable* no Java 17, isso significa que é possível usar *ResultSet* e *Connection* em um bloco try-with-resources para garantir que esses recursos sejam fechados corretamente. No caso do *Connection* os recursos associados que são liberados são relativos à conexão, como sockets de rede e outros recursos de comunicação e para o *ResultSet* representa o conjunto de resultados de uma consulta ao banco de dados como poe exemplo cursores e buffers utilizados para armazenar os resultados da consulta