1. (a) 
\( (1, \text{\texttt{a}}a_ba), (3', \text{\texttt{b}}a_ba), (3, \text{\texttt{a}}b_a_b), (4, \text{\texttt{a}}b_a_b), (5, \text{\texttt{a}}b_a_b), (6, \text{\texttt{b}}a_ba), (7, \text{\texttt{b}}a_ba), (1, \text{\texttt{b}}a_ba), (3', \text{\texttt{a}}b_a_b), (3, \text{\texttt{a}}b_a_b), (h, \text{\texttt{b}}a_ba). \)

(b) 
\( (1, \text{\texttt{a}}ba), (3', \text{\texttt{b}}a_ba), (3, \text{\texttt{a}}b_a_b), (4, \text{\texttt{a}}b_a_b), (4, \text{\texttt{b}}a_ba), (5, \text{\texttt{a}}b_a_b), (6, \text{\texttt{b}}a_ba), (6, \text{\texttt{b}}a_ba), (7, \text{\texttt{b}}a_ba), (1, \text{\texttt{b}}a_ba), (2', \text{\texttt{a}}b_a_b), (2, \text{\texttt{b}}a_ba). \)

(c) 
\( (1, \text{\texttt{a}}ba_ba), (2', \text{\texttt{b}}a_ba_ba), (2, \text{\texttt{b}}a_ba_ba), (4, \text{\texttt{a}}b_a_ba), (4, \text{\texttt{b}}a_ba_ba), (4, \text{\texttt{b}}a_ba_ba), (4, \text{\texttt{a}}b_a_ba), (4, \text{\texttt{b}}a_ba_ba), (5, \text{\texttt{a}}b_a_ba), (6', \text{\texttt{a}}b_a_ba), (6, \text{\texttt{a}}b_a_ba), (7, \text{\texttt{a}}b_a_ba), (7, \text{\texttt{b}}a_ba), (7, \text{\texttt{b}}a_ba), (1, \text{\texttt{b}}a_ba), (3', \text{\texttt{b}}a_ba_ba), (3, \text{\texttt{b}}a_ba_ba), (h, \text{\texttt{b}}a_ba). \)

(d) 
\( (1, \text{\texttt{a}}bba_ba), (3', \text{\texttt{b}}bba_ba), (3, \text{\texttt{a}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (4, \text{\texttt{b}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (5, \text{\texttt{a}}bba_ba), (6', \text{\texttt{a}}bba_ba), (6, \text{\texttt{a}}bba_ba), (7, \text{\texttt{a}}bba_ba), (7, \text{\texttt{b}}bba_ba), (7, \text{\texttt{b}}bba_ba), (1, \text{\texttt{b}}bba_ba), (2', \text{\texttt{a}}bba_ba), (2, \text{\texttt{b}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (5, \text{\texttt{a}}bba_ba), (6', \text{\texttt{a}}bba_ba), (6, \text{\texttt{a}}bba_ba), (7, \text{\texttt{a}}bba_ba), (7, \text{\texttt{b}}bba_ba), (1, \text{\texttt{b}}bba_ba), (2', \text{\texttt{a}}bba_ba), (2, \text{\texttt{b}}bba_ba). \)

(e) 
\( (1, \text{\texttt{a}}bba_ba), (3', \text{\texttt{b}}bba_ba), (3, \text{\texttt{a}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (4, \text{\texttt{b}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (5, \text{\texttt{a}}bba_ba), (6', \text{\texttt{a}}bba_ba), (6, \text{\texttt{a}}bba_ba), (7, \text{\texttt{a}}bba_ba), (7, \text{\texttt{b}}bba_ba), (1, \text{\texttt{b}}bba_ba), (3', \text{\texttt{b}}bba_ba), (3, \text{\texttt{a}}bba_ba), (4, \text{\texttt{a}}bba_ba), (4, \text{\texttt{b}}bba_ba), (5, \text{\texttt{a}}bba_ba), (6', \text{\texttt{a}}bba_ba), (6, \text{\texttt{a}}bba_ba), (6, \text{\texttt{b}}bba_ba). \)

2. Here 1 is the initial state and \( h \) the halting state:

\[
\begin{array}{c|c|c|c|c|c}
1 & a & 1 & c & 1 & a \\
1 & b & 1 & \rightarrow & 1 & b \\
1 & c & 1 & \rightarrow & 1 & c \\
1 & a & h & \rightarrow & 1 & h \\
\end{array}
\]

or

\[
\begin{array}{c|c|c|c|c|c}
1 & a & 1 & 2 & c & 1 \\
1 & b & 1 & \rightarrow & 1 & b \\
1 & c & 1 & \rightarrow & 1 & c \\
1 & a & h & \rightarrow & 1 & h \\
\end{array}
\]

3. The following machine halts on all words in the given language and ‘gets stuck’ on all words that do not belong to the language:

\[
\begin{array}{c|c|c|c|c|c}
1 & a & 2 & \rightarrow & 1 & a \\
2 & a & 1 & \rightarrow & 1 & a \\
\end{array}
\]

It is not hard (but tiresome) to modify this machine to comply with the definition on page 4, Lecture 8. The machine below erases the input word and writes Y if it belongs to the language and N otherwise.
4. The machine reads the word from left to right, simulating the following NFA:

```
1 a 2 →
2 a 1 →
2 b 4 →
3 b 3 ←
3 a 3' ←
3' b 3 ←
3 ' b 5 →
5 a 7 Y
7 Y h Y
4 a 4' ←
4' a 4' ←
4' b 6 →
6 b 8 N
8 N h N
```

Again, this machine halts on the words in the language and ‘gets stuck’ on the remaining words. You can modify it to comply with the definition on page 4, Lecture 9 in the same way as above.

5. This machine is similar to the machine in Question 1: it moves between the first and the last symbol of the current word on the tape; it erases the first and the last symbols if they coincide; if they are different, the machine erases the whole word and writes 0 (or N); if only one symbol is left on the tape, the machine erases it and writes 1 (or Y).

6. The answer is (b). A universal Turing machine takes the program of some Turing machine M and a word \( w \) written on tape, and then in a step-by-step manner simulates the work of \( M \) on the input \( w \). Java Runtime Environment (aka Java Virtual Machine) does a similar thing. It takes a compiled Java code as its input and executes it in a step-by-step manner. Java Compiler just translates Java code into a compact form (suitable for Java Virtual Machine), but does not try to execute its instructions. SQL server is not related to the situation, since SQL queries (unlike Java code) cannot be considered as a Turing machine program.