PREVENÇÃO DE PROBLEMAS

É possível melhorar o código da parte 1:

Observe que muitas coisas dependem da opinião e experiência do desenvolvedor, e por muitas vezes a solução encontrada não é a ideal, assim, alguns pontos no final desse artigo também são passíveis de evolução.

1-) Chave primária da tabela pessoa:

Usamos como Chave primária o campo RG da tabela. Realmente identifica a tabela unicamente, mas pode gerar problemas futuros. É necessário termos uma chave única, que identifica pelo sistema. Assim podemos controlar a tabela unicamente sem depender da abstração que foi feita. Um problema real que pode ocorrer é se nos requisitos do programa o RG não for um atributo desejada para a entidade em questão.
É claro que isso pode ocorrer com qualquer outro campo, mas com nossa chave os problemas são maiores, pois perdemos a identificação dessa tabela no sistema em si.
Enfim, uma nova chave, com número não interessante para que usa o sistema, que fará que o sistema identifique cada entidade independente do que está modulando.
Na prática usaremos uma chave com incremento automático, que nunca pode ser nula:

```
1.  CREATE TABLE pessoa (
2.  id int(4) not null auto_increment,
3.     rg varchar(20) not null,
4.     nome varchar(20) not null,
5.     idade int(2) not null,
6.     cidade varchar(20) not null,
7.     estado varchar(2) not null,
8.     primary key(id)
9.  );
```

O auto_increment quer dizer que a cada pessoa acessada esse número terá seu valor adicionado em um sobre o último valor adicionado. Ou seja, se eu adicionar uma pessoa "José", ele terá o id com valor 1, e adicionar "Maria" logo depois, ela terá o id com

valor 2 e assim segue conforme vamos adicionando novas pessoas.

2-)DAO é DAO!
DAO significa Data Access Object. É um padrão criado pela Sun que prega a separação de acesso aos dados da lógica de negócio (business logic) e disponibiliza uma interface abstrata de persistência a dados independente do banco utilizado. Bem simples e diretamente, sem se aprofundar: temos que ter uma classe de acesso aos dados.

Conforme viram, **acesso**!

Somente com as definições acima é possível ver que quebramos esse padrão na implementação anterior.
Vamos enumerar e falar brevemente o que está fora do padrão no nosso caso.

- Exibir mensagem de erro:

Temos um método que faz a exibição do erro(imprimeErro) que fica na DAO.
A comunicação com o usuário deve ser feita pela View, afinal, esse é o objetivo dessa camada. Conforme já dito, DAO serve para acessar dados, não exibir mensagem.
O melhor a ser feito nesse caso é "passar a bola", ou seja, lançar a exceção. Para isso usamos o comando throw.
Usando throws:

```
1.  public ... throws Exception
```

OBS: Como não tínhamos interface gráfica, foi mantido o método imprimeErro, no outro tutorial, buscando a simplicidade.

E o DAO passa a ser independente de onde é executado, ou seja, WEB ou desktop podem usar. Antigamente, somente em casos Desktop(JavaSwing) iríamos ver a mensagem(JOptionPane), com throws podemos delegar o tratamento de exceção para quem usa o DAO.

- Conectar ao banco de dados:

DAO não tem nada a ver com a conexão, a conexão deve ser papel de outra classe, DAO deve permitir a manipulação dos dados, não deve se "envolver" com a impressão de dados e nem com a conexão. Outra coisa, com informações locais da conexão faz nosso DAO perder a **portabilidade e flexibilidade**. No nosso caso reformulamos a conexão para conseguir maior flexibilidade. Como já dito, o resultado obtido não é o perfeito, muitos podem opinar e melhorar esse resultado!

Por fim, o DAO deve o mais genérico possível! Em alguns pontos omitimos essa abstração do DAO para conseguir maior simplicidade na hora de expor nosso objetivo, mas tenha em mente sempre deixar as classes DAO flexíveis na medida do possível, ou seja, o que poder ser feito para melhorar a abstração, FAÇA!

3-) Código

O código deixa muito a desejar em alguns pontos. Sem muito papo, vamos a eles:

-Uso de Statement
Absolutamente não está errado, mas o Statement deixa o código sujo, não ajuda na manutenção e nem na visibilidade. Uma outra opção é o PrepareStatement, onde:

*****Não precisamos imediatamente colocar as aspas simples que representam String, por exemplo, ele cuida da montagem do SQL.

*****Não é necessário a concatenação da String do comando com os valores.
Com somente esses dois argumentos fica claro que o uso de PrepareStatement ao contrário de Statement é mais vantajoso.

-Tipo concreto de List invés de tipo abstrato.

No retorno do método estávamos usando Vector, um tipo de List. O melhor seria usar o tipo genérico, List, pois no garante maior flexibilidade, conforme já foi dito anteriormente. Cada tipo concreto de List se aplica melhor em determinados casos. Tendo uma List genérica você pode utilizar o resultado da forma que melhor se aplica a sua necessidade.