



**UNIVERSIDADE DE BRASÍLIA**  
**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**PROJETO FINAL DA DISCIPLINA**  
**BANCO DE DADOS**

Gabriel Rocha Fontenele – 15/0126760

Gabriel Teixeira Ribeiro – 15/0126891

Marcos Emanuel de Farias – 16/0052882

Otávio Souza de Oliveira – 15/0143401

**Brasília – DF**

**2019/1**

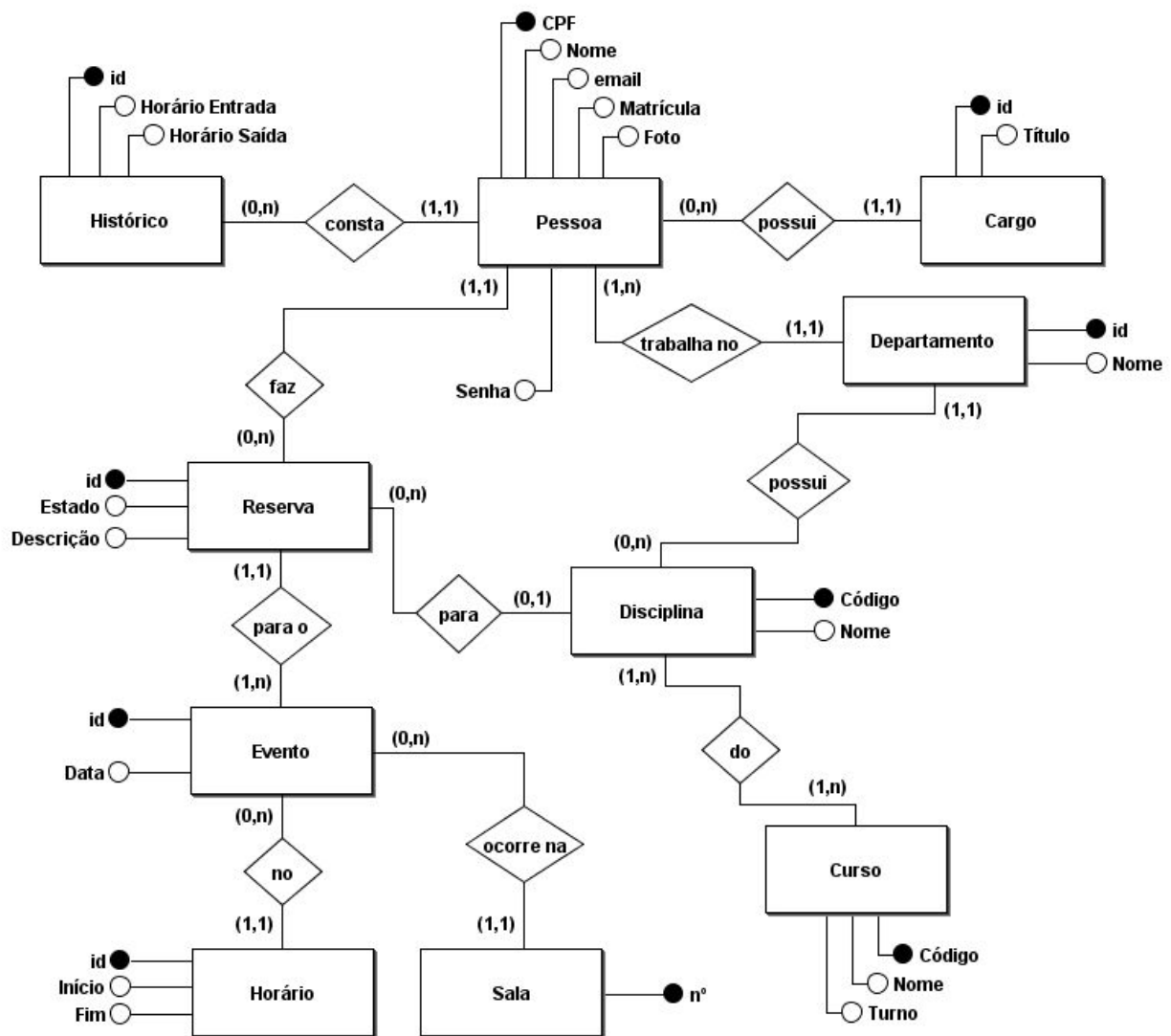
## **1.INTRODUÇÃO**

O projeto realizado e descrito a seguir consiste na descrição de um modelo de banco de dados para uma aplicação que o implementa através dos conceitos de criação, modelagem e gerenciamento de banco de dados.

O aplicativo em questão visa um modelo que tem por objetivo catalogar a entrada e saída de pessoas (alunos, professores, funcionários) e organizar a reserva de salas pelos professores no Laboratório de Informática da UnB (LINF). Neste modelo, as reservas podem ainda se caracterizar como pontuais (para realização de eventos, ou aulas de disciplinas que normalmente são ministradas em outras salas) ou como contínuas (para aulas que serão ministradas no LINF durante todo o semestre).

O repositório com todos os códigos e modelos utilizados no projeto estão em:  
<https://github.com/moltentheory/LINFadmin>

## 2. DIAGRAMA DE ENTIDADE RELACIONAMENTO

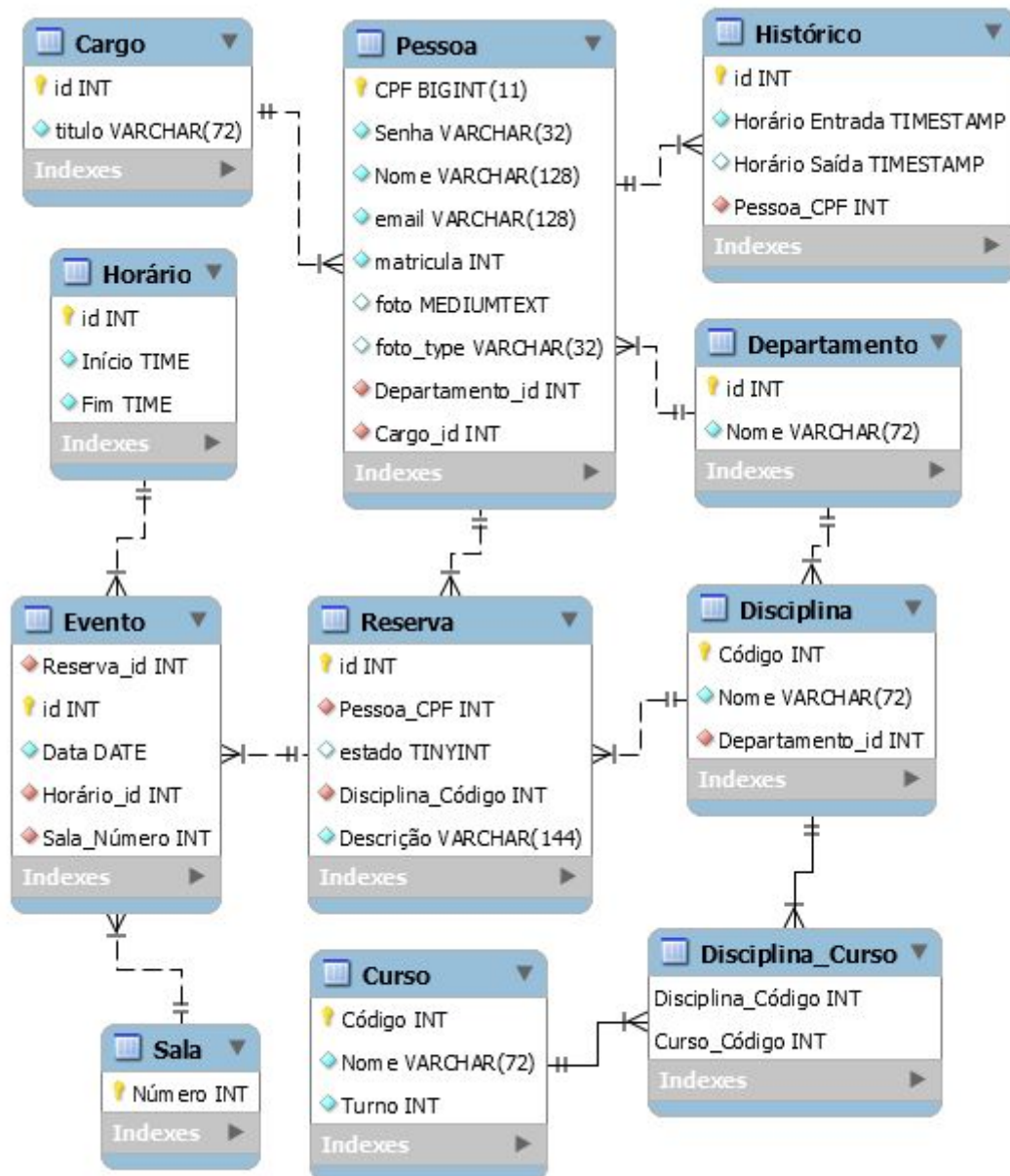


Os atributos estão são representados pelos círculos, em que os preenchidos são os identificadores. Algumas das entidades tem um motivo especial para sua existência.

A entidade *Cargo* é necessária para categorizar a função exercida por uma certa pessoa em um determinado departamento. Assim, podemos criar permissão (professores podem fazer reserva, alunos não) e prioridade de reserva a usuários de certos departamentos.

A existência da entidade *Evento* se dá pela possibilidade da realização de uma série de eventos em uma *Reserva*. Isso configura uma relação consistente para a existência de reservas contínuas - como, por exemplo, a reserva de todas as aulas de uma disciplina para um semestre.

### 3. MODELO RELACIONAL



As tabelas do modo relacional caracterizam-se por dar ênfase aos dados apresentados por cada tabela, ou seja, os atributos em uma entidade, e como se apresentam as relações na descrição deste modelo.

Chaves amarelas são chaves primárias. As vermelhas são chaves estrangeiras. Os dados sem o sinal de chave são na verdade chaves estrangeiras e primárias para a tabela em questão.

#### 4. CONSULTAS EM ÁLGEBRA RELACIONAL

Listadas abaixo estão cinco consultas que retornam informações que o usuário pode precisar com frequência, pensadas para aprimorar o sistema e disponibilizadas na barra de pesquisa do **CRUD**, local de fácil acesso. Todas as consultas fornecem uma espécie de histórico ou relatório envolvendo três ou mais entidades do projeto. O primeiro passo para a implementação das consultas foi escrevê-las na forma de álgebra relacional.

- Lista de eventos das próximas duas semanas, incluindo horários do evento, nome e cargo da pessoa que fez a reserva.
  - $\pi$  sala\_numero, dia, inicio, fim, nome, titulo, descricao ( $\sigma$  (EVENTO.horario\_id = HORARIO.id) AND (EVENTO.reserva\_id = RESERVA.id) AND (pessoa\_cpf = cpf) AND (cargo\_id = CARGO.id) AND (dia < Data de Hoje + 14 dias) (EVENTO x HORARIO x RESERVA x PESSOA x CARGO))
- Lista dos eventos reservados para o departamento de Ciência da Computação
  - $\pi$  sala\_numero, dia, inicio, fim, descricao ( $\sigma$  (EVENTO.horario\_id = HORARIO.id) AND (EVENTO.id = RESERVA.id) AND (disciplina\_codigo = DISCIPLINA.codigo) AND (departamento\_id = DEPARTAMENTO.id) AND (nome = "Departamento de Ciência da Computação) (EVENTO x HORARIO x RESERVA x DISCIPLINA x DEPARTAMENTO))
- Ficha de todos os professores que constam no histórico
  - $\pi$  nome, email, foto, DEPARTAMENTO.nome, horario\_entrada ( $\sigma$  (cpf = pessoa\_cpf) AND (DEPARTAMENTO.id = departamento\_id) AND (CARGO.id = cargo\_id) AND (titulo = "Professor") (PESSOA x CARGO x DEPARTAMENTO x HISTÓRICO))
- Histórico do último mês, contendo todas as informações pessoais dos que nele constam
  - $\pi$  horario\_entrada, horario\_saida, titulo, matricula, nome, email, foto, DEPARTAMENTO.nome ( $\sigma$  (cpf = pessoa\_cpf) AND (DEPARTAMENTO.id = departamento\_id) AND (CARGO.id = cargo\_id) AND (horario\_entrada > Data de hoje - 31 dias) (PESSOA x CARGO x DEPARTAMENTO x HISTÓRICO))


- Ficha geral de reservas, incluindo salas, data e hora dos eventos, disciplinas, departamento e nome da pessoa que fez a reserva.
- $\pi$  sala\_numero, dia, inicio, fim, descricao, DISCIPLINA.nome, DEPARTAMENTO.nome, PESSOA.nome  $(\sigma_{(RESERVA.id = EVENTO.id) \text{ AND } (pessoa\_cpf = cpf) \text{ AND } (disciplina\_codigo = DISCIPLINA.codigo) \text{ AND } (horario\_id = HORARIO.id) \text{ AND } (DISCIPLINA.departamento\_id = DEPARTAMENTO.id)} (RESERVA \times EVENTO \times HORARIO \times DISCIPLINA \times DEPARTAMENTO \times PESSOA))$

## 5. AVALIAÇÃO DAS FORMAS NORMAIS

A normalização visa eliminar redundâncias e relações, dentro de uma tabela, que causem problemas de inserção, remoção ou atualização dos dados.

Para a primeira forma normal (**1FN**), devemos garantir que o valor das colunas de uma tabela é indivisível. Assim sendo, não devem existir atributos multivalorados (em MER) ou redundâncias (repetição de valores de uma coluna em várias linhas da tabela). Em seguida, para a segunda forma normal (**2FN**), devemos criar novas tabelas para grupos de dados que se aplicam a vários registros e relacioná-las através de chaves estrangeiras. Por fim, devemos separar os atributos dependentes de forma transitiva da chave primária em novas tabelas para a terceira forma normal (**3FN**).

A conversão do MER para o MR por si só já é um forte indicador que considera o esquema relacional normalizado. Entretanto, avaliaremos a normalização para as tabelas mais importantes do projeto, a fim de garantir que o esquema é válido. Começaremos utilizando como exemplo o seguinte registro:

RESERVA				
13		Aprovada		
Professor:	Maristela Terto	CPF:	000.000.000-01	
Email:	<a href="mailto:maristela@unb.com">maristela@unb.com</a>	Matrícula	555555	
	Código Disciplina:		116378	
	Disciplina:		Banco de Dados	
	EVENTO			DESCRIÇÃO
	Nº	Sala	Data	Laboratório de SQL.
	37	2	18/06/2019	
	38	3	20/06/2019	
	39	2	25/06/2019	
	40	2	27/06/2019	

Vamos transcrever os títulos de cada campo para encontrarmos quais campos se repetem em uma mesma tabela (ainda que com dados diferentes):

***{Número da reserva, Estado da reserva, Professor, CPF, Email, Matrícula, Foto, Código da disciplina, Nome da Disciplina, {Número do Evento}, {Sala}, {Data}, Descrição}***

Mantendo a ordem, vamos renomear os campos para termos uma maior aproximação com o MR do projeto. Assim, os campos acima podem ser reescritos, respectivamente, através do conjunto: ***{Reserva\_id, Estado, Pessoa\_Nome, CPF, Email, Matrícula, Foto, Disciplina\_Código, Disciplina\_Nome, {Evento\_id}, {Sala}, {Data}, Descrição}***

O atributo *Reserva\_id* é notavelmente o atributo que melhor identifica um registro deste tipo. Consideramos então que ele é a nossa primeira PK. Observamos então que uma única tabela para cada *Reserva\_id* incluiria redundância de campos, com colunas divididas por mais de um valor. Inferimos que, se tomarmos *Evento\_id* como nossa próxima PK, eliminaremos esse problema: ***{Reserva\_id, Evento\_id}; {Estado, CPF, Pessoa\_Nome, Email, Matrícula, Foto, Disciplina\_Código, Disciplina\_Nome, Sala, Data, Descrição}***

Registro	
Reserva_id	<PK>
Estado	
CPF	
Pessoa_Nome	
Email	
Matrícula	
Foto	
Disciplina_Código	
Disciplina_Nome	
Evento_id	<PK>
Sala	
Data	
Descrição	

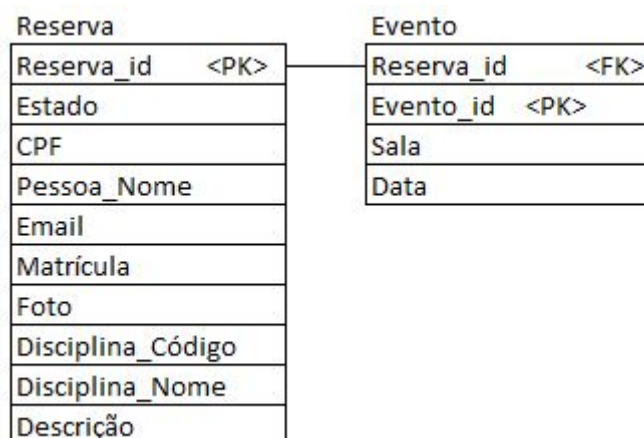
Chegamos, assim, na 1FN. Para a 2FN, utilizaremos os atributos-chave para descobrir de quem os atributos não-chaves são dependentes. Sendo assim, temos que:

***{Reserva\_id} → {Estado, CPF, Pessoa\_Nome, Email, Matrícula, Foto, Disciplina\_Código, Disciplina\_Nome, Descrição}***

Pelo registro, observamos a existência de vários *Evento\_id* em uma única *Reserva\_id*. Sendo assim, concluímos que um evento faz referência a uma certa reserva.

***{Evento\_id} → {Sala, Data, Reserva\_id}***

***{Reserva\_id, Evento\_id} → { }***

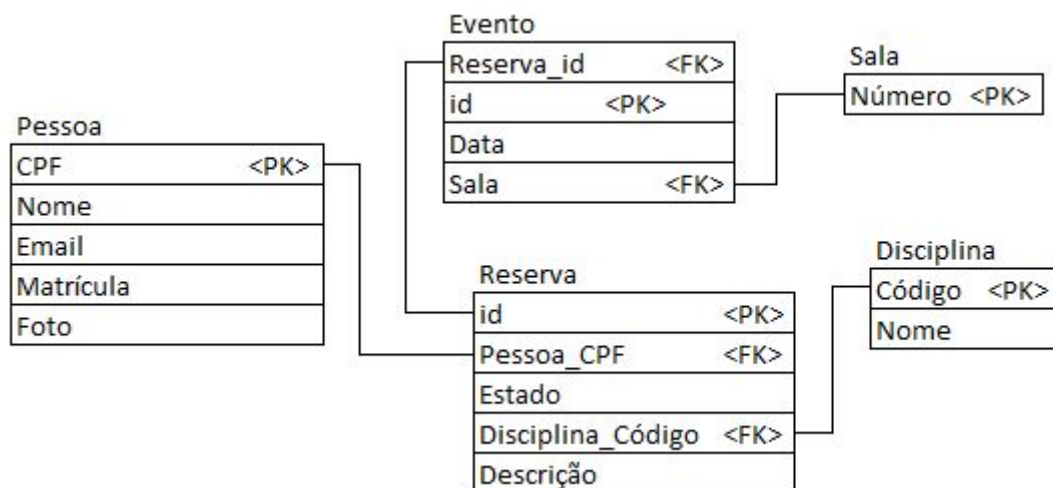




Nossas tabelas agora estão em 2FN. Avaliaremos se existe dependência transitiva interna dentro de cada tabela. Para isso, enumeremos os dados, ignorando as chaves primárias e ligando os atributos que estabelecem alguma relação de dependência entre si.

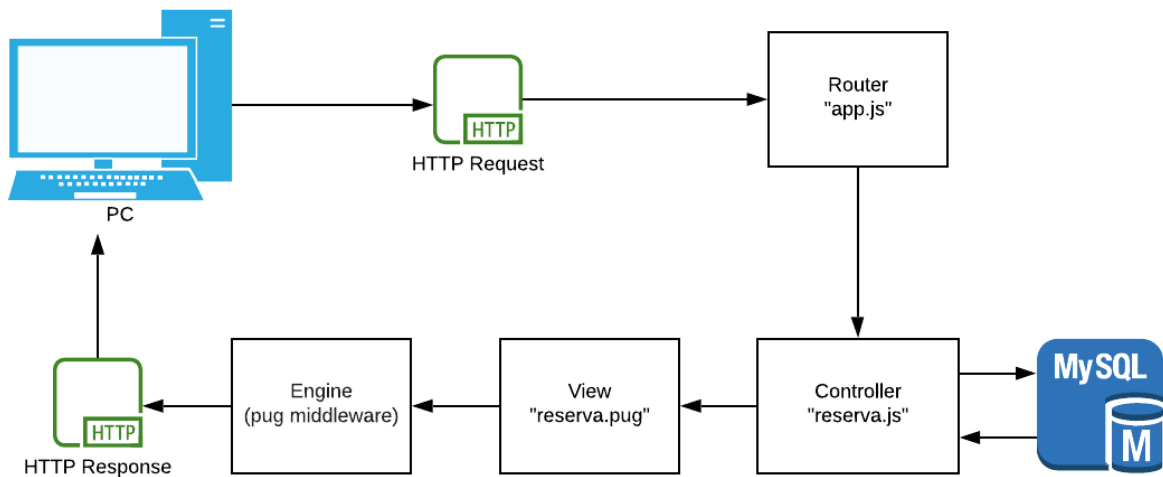
(~~Reserva\_id~~, Estado, CPF, Pessoa\_Nome, Email, Matrículo, Foto, Disciplina\_Código, Disciplina\_Nome, Descrição)  
 (~~Evento\_id~~, Sala, Data, Reserva\_id)

Vamos lembrar também que as salas disponíveis existem independentemente das reservas registradas, assim:



Chegamos então a essa versão final da avaliação, resultando em cinco tabelas para entidades distintas e uma tabela para relacionar as entidades *Reserva* e *Disciplina*.

## 6. DIAGRAMA DA CAMADA DE MAPEAMENTO



### Processos da camada de mapeamento:

- **Usuário(PC):** O usuário do sistema ao acessar uma página, ou ao tentar fazer uma reserva, cadastramento; envia um request HTTP que possui um rota(Router) no sistema.
- **Router:** Ao receber o request do usuário, a rota é quem direciona a uma controladora(Controller) que é responsável pela ação requerida pelo usuário.
- **Controller:** A controladora baseada na rota do request será responsável por enviar a query ao banco de dados, e o banco vai retornar o resultado da query, e após obter o resultado a controladora vai gerar a view apropriada para mostrar o resultado obtido.
- **View:** A view gerada pela controladora vai mostrar os resultados obtidos a partir da controladora, porém antes disso a view com extensão *“.pug”* passa por um *Engine(pug middleware)* para então renderizar uma requisição HTTP com formato HTML na tela do usuário.

## 7. PROCEDURE

```
CREATE DEFINER='root'@'localhost' PROCEDURE `aprova_reserva`()
BEGIN

CREATE TEMPORARY TABLE reserva_evento (rid int, eid int, estado int);
INSERT INTO reserva_evento
    SELECT r.id AS rid, e.id AS eid, r.estado AS estado
    FROM evento e
    INNER JOIN reserva r
    ON r.id = e.reserva_id;

UPDATE reserva res
SET res.estado =
    CASE WHEN
        (
            (
                SELECT COUNT(id)
                FROM evento
                WHERE reserva_id = res.id
            ) = (
                SELECT COUNT(e1.id)
                FROM evento e1
                INNER JOIN evento e2
                ON e1.dia = e2.dia
                AND e1.horario_id = e2.horario_id
                AND e1.sala_numero = e2.sala_numero
                INNER JOIN reserva_evento re
                ON re.eid = e2.id
                WHERE e1.reserva_id = res.id
                AND re.estado != 1
            )
        )
    THEN 2
    ELSE 1
    END
WHERE res.estado = 0;
DROP TABLE IF EXISTS reserva_evento;
END
```

A procedimento armazenado no banco de dados, escrito através do MySQL, cria uma tabela temporária que compara o número de eventos registrados, por uma *r1*, consigo mesmo adicionado do número de eventos, registrados por *r2* - sendo *r1* e *r2*, possíveis reservas que esperam aprovação e contém eventos que disputam por uma mesma *data*, *horário* e *número de sala*. Ainda é definido que a tabela para comparação será formada apenas por eventos de reservas que já foram aprovadas ou estão esperando por aprovação.

Isso se torna possível aos conferir o valor atribuído a *estado*, na tabela *Reserva*. Se o valor for '0', a reserva está esperando por aprovação. Se for '1', a reserva já foi rejeitada. Já '2', significa que a transação já foi aprovada e, portanto, sempre terá prioridade sobre as novas requisições de reserva.

Por fim, o procedimento realiza um teste condicional para definir se as reservas que disputam *DHS* (dia, data e sala iguais) serão aprovadas. Todas as reservas que não apresentam tal conflito, são aprovadas. Todas as demais serão diretamente rejeitadas, seja as que ainda não foram avaliadas ou as que disputam *DHS* com reservas já aprovadas.

Esse procedimento é chamado pelo evento a seguir e executado todos os dias, uma vez por dia.

```
CREATE EVENT IF NOT EXISTS `Aprovação de Reservas`  
ON SCHEDULE  
  EVERY 24 DAY_HOUR  
COMMENT 'Aprovando ou rejeitando reservas!'  
DO  
  CALL linf.aprova_reserva();
```