Para evitar a anomalia de Lost Update no sistema de venda de ingressos, especialmente em situações de alta concorrência, podemos modelar cada ingresso de um show como um registro individual em uma tabela, em vez de utilizar um contador de ingressos disponíveis.

Nesse novo modelo, temos três tabelas principais:

- `user`: que armazena os compradores;
- `show`: que representa os eventos;
- `ticket`: onde cada linha corresponde a um ingresso específico, vinculado a um show e, opcionalmente, a um comprador.

A lógica de venda consiste em selecionar os ingressos disponíveis, ou seja, que satisfazem `buyer_id IS NULL`, utilizando a cláusula `FOR UPDATE SKIP LOCKED`. Essa cláusula bloqueia os registros selecionados para evitar que outras transações os modifiquem simultaneamente, e ignora aqueles que já estão bloqueados, permitindo o processamento concorrente sem conflitos. Em seguida, os ingressos retornados são atualizados, preenchendo o campo `buyer_id` com o identificador do comprador. Se a quantidade de ingressos modificados for inferior à quantidade solicitada, a transação é cancelada via `ROLLBACK` para evitar compras parciais. Caso contrário, a transação é efetivada por meio do `COMMIT`.

Essa abordagem elimina a necessidade de serializar todas as transações (ou seja, não é preciso fazer um `SELECT ... FOR UPDATE` na tabela do evento inteiro), pois cada transação apenas bloqueia as linhas dos ingressos que está reservando. Assim, múltiplas transações podem ocorrer de forma concorrente, sem correr o risco de vender mais ingressos do que o disponível, e sem causar contenção excessiva, pois cada transação atua apenas sobre os ingressos efetivamente reservados por ela.

Script das tabelas

```sql
CREATE DATABASE ticket_system;
USE ticket_system;

CREATE TABLE `user`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE `show`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE `ticket`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT,
    `show_id` INTEGER NOT NULL,
    `buyer_id` INTEGER NULL,

    FOREIGN KEY (`show_id`) REFERENCES `show`(`id`),
    FOREIGN KEY (`buyer_id`) REFERENCES `user`(`id`)
);
```

Script da transação

```
CREATE PROCEDURE reserve_tickets(
    IN p_show_id INT,
    IN p_buyer_id INT,
    IN p_amount INT
)
BEGIN
    START TRANSACTION;

    UPDATE `ticket` SET `buyer_id` = p_buyer_id WHERE `id` IN (
        SELECT id FROM (
            SELECT `id` FROM `ticket`
            WHERE `show_id` = p_show_id AND `buyer_id` IS NULL
            LIMIT p_amount
            FOR UPDATE SKIP LOCKED
        ) AS t
    );

    IF ROW_COUNT() = p_amount THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
END
```

Scripts para execução do exemplo

```
DROP DATABASE IF EXISTS ticket_system;
CREATE DATABASE ticket_system;

USE ticket_system;

CREATE TABLE `user`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE `show`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE `ticket`(
    `id` INTEGER PRIMARY KEY AUTO_INCREMENT,
    `show_id` INTEGER NOT NULL,
    `buyer_id` INTEGER NULL,

    FOREIGN KEY (`show_id`) REFERENCES `show`(`id`),
    FOREIGN KEY (`buyer_id`) REFERENCES `user`(`id`)
);

INSERT INTO `user` VALUES (); -- id = 1
INSERT INTO `user` VALUES (); -- id = 2
```

```
INSERT INTO `show` VALUES (); -- id = 1

INSERT INTO `ticket` (`show_id`, `buyer_id`) VALUES (1, NULL);
INSERT INTO `ticket` (`show_id`, `buyer_id`) VALUES (1, NULL);
INSERT INTO `ticket` (`show_id`, `buyer_id`) VALUES (1, NULL);
INSERT INTO `ticket` (`show_id`, `buyer_id`) VALUES (1, NULL);
INSERT INTO `ticket` (`show_id`, `buyer_id`) VALUES (1, NULL);
```

```
USE ticket_system;

DROP PROCEDURE IF EXISTS reserve_tickets1;

CREATE PROCEDURE reserve_tickets1(
    IN p_show_id INT,
    IN p_buyer_id INT,
    IN p_amount INT
)
BEGIN
    START TRANSACTION;

    UPDATE `ticket` SET `buyer_id` = p_buyer_id WHERE `id` IN (
        SELECT id FROM (
            SELECT `id` FROM `ticket`
            WHERE `show_id` = p_show_id AND `buyer_id` IS NULL
            LIMIT p_amount
            FOR UPDATE SKIP LOCKED
        ) AS t
    );

    IF ROW_COUNT() = p_amount THEN
        SELECT SLEEP(10);
        COMMIT;
    ELSE
        SELECT SLEEP(10);
        ROLLBACK;
    END IF;
END
```

```
USE ticket_system;

DROP PROCEDURE IF EXISTS reserve_tickets2;

CREATE PROCEDURE reserve_tickets2(
    IN p_show_id INT,
    IN p_buyer_id INT,
    IN p_amount INT
)
BEGIN
```

```
    START TRANSACTION;

    UPDATE `ticket` SET `buyer_id` = p_buyer_id WHERE `id` IN (
        SELECT id FROM (
            SELECT `id` FROM `ticket`
            WHERE `show_id` = p_show_id AND `buyer_id` IS NULL
            LIMIT p_amount
            FOR UPDATE SKIP LOCKED
        ) AS t
    );

    IF ROW_COUNT() = p_amount THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
END
```

```
USE DATABASE ticket_system;

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

CALL reserve_tickets1(1, 1, 3); -- reserva 3 tickets para o usuário 1 no show 1
(commit)
```

```
USE DATABASE ticket_system;

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

CALL reserve_tickets2(1, 1, 3); -- reserva 3 tickets para o usuário 2 no show 1
(rollback)
CALL reserve_tickets2(1, 2, 2); -- reserva 2 tickets para o usuário 2 no show 1
(commit)
```