Import System Schema Proposal

Overview

This document proposes database schema extensions to support HitTrax + Sensor (Blast/Diamond Kinetics) import functionality.

Existing Schema Analysis

✅ Already Exists

- **HitTraxSession**: Container for HitTrax session data
- **HitTraxEvent**: Individual pitch/swing events from HitTrax
- **SwingAnalysis**: Analysis results for video swings
- **SwingMetrics**: Detailed biomechanical metrics

❌ Missing (Proposed New Models)

Proposed Schema Extensions

1. ImportSession Model

Purpose: Track each import operation (per-player or bulk)

```
model ImportSession {
  id                 String   @id @default(uuid())
  userId             String   // Coach/admin who initiated import
  user               User     @relation("UserImportSessions", fields: [userId], refer-
ences: [id])

  importType         String   // "per_player" | "bulk"
  sourceTypes        String[] // ["hittrax", "blast", "dk"]

  // File tracking
  fileNames          String[] @db.Text
  fileCount          Int      @default(0)

  // Statistics
  totalSwings        Int      @default(0)
  matchedSwings      Int      @default(0)
  unmatchedSwings    Int      @default(0)
  playersDetected    Int      @default(0)

  status             String   @default("processing") // "processing", "completed", "fai
led"
  errorMessage       String?  @db.Text

  // Links to created data
  hittraxSessions    HitTraxSession[] @relation("ImportHitTraxSessions")
  sensorSwings       SensorSwing[]    @relation("ImportSensorSwings")

  createdAt          DateTime @default(now())
  completedAt        DateTime?

  @@map("import_sessions")
  @@index([userId])
  @@index([createdAt])
}
```

2. SensorSwing Model

Purpose: Store bat sensor data (Blast, Diamond Kinetics)

```
model SensorSwing {
  id                 String   @id @default(uuid())

  // Player assignment
  userId             String?
  user               User?    @relation("UserSensorSwings", fields: [userId], refer-
ences: [id])

  // Session context
  importSessionId    String?
  importSession      ImportSession? @relation("ImportSensorSwings", fields: [importSes-
sionId], references: [id])

  // Sensor metadata
  sensorType         String   // "blast" | "diamond_kinetics"
  sensorDeviceId     String?  // Device serial/ID
  sensorTimestamp    DateTime // Original sensor timestamp

  // Matching to HitTrax
  hittraxEventId     String?  @unique
  hittraxEvent       HitTraxEvent? @relation(fields: [hittraxEventId], references:
[id])
  matchTimeDelta     Float?   // Milliseconds difference from match (if matched)

  // Core sensor metrics
  batSpeed           Float?   // mph
  attackAngle        Float?   // degrees
  timeToContact      Float?   // milliseconds
  peakHandSpeed      Float?   // mph

  // Blast-specific metrics
  blastFactor        Float?   // Blast proprietary metric
  powerOutput        Float?   // Watts

  // DK-specific metrics
  rotationMetric     Float?   // DK rotation score

  // Raw data storage (for future use)
  rawDataJson        Json?    // Full sensor payload

  // Assignment status
  assigned           Boolean  @default(false)
  assignedAt         DateTime?
  assignedBy         String?  // Admin who assigned

  createdAt          DateTime @default(now())

  @@map("sensor_swings")
  @@index([userId])
  @@index([sensorTimestamp])
  @@index([assigned])
  @@index([sensorType])
}
```

3. Schema Modifications

Add to HitTraxSession:

```
// Add relation to ImportSession
importSessionId   String?
importSession     ImportSession? @relation("ImportHitTraxSessions", fields: [im-
portSessionId], references: [id])
```

Add to HitTraxEvent:

```
// Add relation to SensorSwing
sensorSwing       SensorSwing?  // One-to-one via sensorSwing.hittraxEventId
```

Add to User:

```
// Add new relations
importSessions    ImportSession[]  @relation("UserImportSessions")
sensorSwings      SensorSwing[]    @relation("UserSensorSwings")
hittraxSessions   HitTraxSession[] @relation("UserHitTraxSessions") // Already exists
```

Migration Strategy

1. **Phase 1**: Add new models (ImportSession, SensorSwing)
2. **Phase 2**: Add relations to existing models
3. **Phase 3**: Test with sample data
4. **Phase 4**: Deploy to production

Data Flow

Per-Player Import

```
1. Upload CSV(s) → S3
2. Create ImportSession (importType="per_player")
3. Parse HitTrax CSV → Create HitTraxSession + HitTraxEvents
4. Parse Sensor CSV → Create SensorSwings (with userId)
5. Match by timestamp → Link via hittraxEventId
6. Update ImportSession statistics
7. Return summary to UI
```

Bulk Import

```
1. Upload CSV(s) or ZIP → S3
2. Create ImportSession (importType="bulk")
3. Parse all files
4. Detect players by:
   - HitTrax batter name
   - Sensor device/player name
   - File naming patterns
5. For each detected player:
   - Create HitTraxSession + HitTraxEvents
   - Create SensorSwings
   - Match by timestamp
6. Flag unmatched/unassigned swings
7. Update ImportSession statistics
8. Return summary with unassigned list
```

Query Patterns

Get all unassigned sensor swings

```
await prisma.sensorSwing.findMany({
  where: { assigned: false },
  include: { importSession: true },
  orderBy: { createdAt: 'desc' }
});
```

Get matched swings for a player

```
await prisma.hitTraxEvent.findMany({
  where: {
    session: { userId: playerId },
    sensorSwing: { isNot: null }
  },
  include: { sensorSwing: true }
});
```

Get import session summary

```
await prisma.importSession.findUnique({
  where: { id: importId },
  include: {
    hittraxSessions: { include: { events: { include: { sensorSwing: true } } } },
    sensorSwings: true,
    user: { select: { name: true, email: true } }
  }
});
```

Benefits

1. **Traceability**: Every import tracked with metadata
2. **Flexibility**: Support multiple sensor types
3. **Matching**: Timestamp-based linking preserves both datasets

4. **Bulk Operations**: Handle multiple players/sessions
5. **Admin Tools**: Easy to find and assign unmatched data
6. **Future-Proof**: JSON storage for raw sensor data

Next Steps

1. ✅ Create this proposal
2. ⏳ Get approval/feedback
3. ⏳ Implement Prisma migrations
4. ⏳ Build TypeScript types
5. ⏳ Implement import logic