数据库设计文档 V0.1

创建者	z yx
创建时间	@2025年10月30日 22:21
类别	Datebase TechSpec
上次编辑者	z yx
上次更新时间	@2025年10月30日 23:00
状态	进行中

好的！下面给出一套基于 **Isar** 的本地数据库架构，并内置“可与 Notion 双向同步”的字段与队列设计。字段/实体命名与您的页面规范完全对齐：**Sentence / Highlight / ReadingPrefs / SyncQueue**，并补充 **Notion 配置**（Token、Database 绑定）。文末含建议的 Notion 数据库字段映射与同步流程要点。

一、核心实体（Isar）

> 依据「单词学习页」的数据模型（Sentence/Highlight/ReadingPrefs）与行为边界（删除、跨行高亮、句级熟悉度）【】【】，以及产品 PRD 的本地优先+成功后标记已同步的模式与必要元字段（updatedAtLocal/updatedAtRemote/syncState、以 Notion page_id 为主键）【】【】。

```
// Dart / Isar 模型（示例）
// ——— Common ———
enum SyncState { synced, pending, conflict, failed }
enum FamiliarState { familiar, unfamiliar, neutral }

// ——— Sentence ———
@collection
class Sentence {
  Id id = Isar.autoIncrement;        // 本地自增主键
  String? notionPageId;              // Notion page_id（远端主键）
```

```
  String text;
  @enumerated
  FamiliarState familiarState;
  DateTime createdAt;                // 句子创建时间（本地记录）
  DateTime? deletedAt;               // 逻辑删除支持（与规范一致）

  // 同步元数据（PRD 建议）
  DateTime updatedAtLocal;           // 最近本地编辑时间
  DateTime? updatedAtRemote;         // 最近远端编辑时间（Notion last_edited_time）
  @enumerated
  SyncState syncState;

  // 索引建议
  // index on notionPageId(unique), updatedAtLocal, updatedAtRemote, deletedAt
}

// ——— Highlight ———
@collection
class Highlight {
  Id id = Isar.autoIncrement;
  String? notionPageId;              // 若高亮独立存为 Notion 子库，则同样保存 page_id
  int sentenceLocalId;               // 关联本地 Sentence.id
  String? sentenceNotionPageId;      // 关联远端 Sentence.page_id（用于直连映射）
  int start;                         // 以字符索引存储（跨行/跨词）【规范要求】
  int end;
  String color;                      // "yellow"|"pink"|"blue"|"green"
  String? note;                      // 0..1 条简注
  DateTime updatedAtLocal;
  DateTime? updatedAtRemote;
  @enumerated
  SyncState syncState;
  DateTime? deletedAt;

  // 索引建议
```

```
  // composite index: (sentenceLocalId, start, end), plus updatedAtLocal, de
letedAt
}

// ——— ReadingPrefs（全局仅一条，或按 profile 扩展） ———
@collection
class ReadingPrefs {
  Id id = 1;                        // 固定单例
  String theme;                     // "light"|"sepia"|"dark"
  int fontSize;                     // 14..22
  double lineHeight;                // 1.4..2.0
  int paragraphSpacing;             // 4..16

  DateTime updatedAtLocal;
  DateTime? updatedAtRemote;
  @enumerated
  SyncState syncState;
}
```

- 上述字段覆盖 UI 规范中的 **句级熟悉/不熟** 存储、**跨行/跨词** 高亮存储、**删除的二级确认/逻辑删除** 等关键约束【】【】。
- **ReadingPrefs** 与欢迎页“持久化偏好到本地或 Notion”一致【】。

二、同步队列（Isar）

> 参照“同步队列页”的数据结构与状态机（Pending/Syncing/Success/Failed/...）、错误码与退避策略等【】【】。

```
@collection
class SyncQueueItem {
  Id id = Isar.autoIncrement;
  String queueId;                   // "sq_<uuid>" 用于前端引用
  String entityType;                // "sentence"|"highlight"|"pref"
  String op;                        // "create"|"update"|"delete"
  String entityLocalKey;            // 本地主键/复合键（如 sentenceLocalId 或
highlightId）
```

```
  String? entityNotionPageId;         // 远端 ID（已存在时）
  Map<String, dynamic> payload;       // 差异或全量 JSON（序列化为 string
存）
  String status;                      // "pending"|"syncing"|"success"|"failed"|"can
celed"|"skipped"
  int attempt;                        // 已重试次数
  int maxAttempt;                     // 默认 5
  String? lastErrorCode;              // "HTTP_401"|"NET_TIMEOUT"|...
  String? lastErrorMessage;
  DateTime? lastErrorAt;
  Map<String, dynamic>? lastErrorMeta; // requestId, endpoint, duration_m
s...
  DateTime createdAt;
  DateTime updatedAt;
  int priority;                       // 调度优先级（默认 0）

  // 索引建议：status, updatedAt, entityType+entityLocalKey, priority
}
```

- 队列项来源：对 **Sentence/Highlight/ReadingPrefs** 的 create/update/delete 操作都入队一条记录【】。
- 失败日志需要能回放“请求/响应片段、Notion/本地对象 ID 对照”，因此保留 **lastErrorMeta** 等字段以便详情面板展示【】。

三、Notion 配置（Isar）

> 设置页流程：Token → 测试 → 绑定数据库（串行依赖），并持久化 Token/Database 解析/绑定结果【】【】。数据结构建议已在规范中给出【】。

```
@collection
class NotionAuth {
  Id id = 1;
  String token;                       // 加密后保存（flutter_secure_storage 持 Toke
n）【PRD】
  String status;                      // "untested"|"success"|"error"
  DateTime? testedAt;
```

```
  String? errorMessage;
}

@collection
class NotionDatabaseBinding {
  Id id = 1;
  String rawUrl;                  // 用户粘贴的 URL（可为空，若直接给 database
_id）
  String databaseId;              // 32 位 ID
  String databaseName;
  String? workspace;
  DateTime boundAt;
}
```

> 对应的 UI/交互要求与状态字段，已在设置页规范中明确（测试成功/失败、解析 database_id、重置/解绑）【】【】【】。

四、索引与约束

- **Sentence**：`notionPageId`（唯一）、`updatedAtLocal`（倒序列表渲染/同步扫描）、`deletedAt`（过滤）。
- **Highlight**：`(sentenceLocalId, start, end)` 复合唯一索引，防止重复块；`updatedAtLocal`；`deletedAt`。
- **ReadingPrefs**：单例 `id=1`；如未来支持多配置，可引入 `profileId`。
- **SyncQueueItem**：`status`（过滤批量操作）、`updatedAt`、`priority`、`entityType+entityLocalKey`。

五、与 Notion 的字段映射（建议）

> PRD 建议以 Notion 数据库为中心，使用 page_id 作为远端主键，且本地保存远端/本地更新时间与同步态【】。

1) Notion 数据库（建议拆为 3 个 DB）

https://www.notion.so/app-29adc335259a806488d1c359a12783dc?source=copy_link#29adc335259a803694a1e090d6d4423e

- **DB_Sentences**
  - Title（title）：`text`（可按句子前若干字作为标题，其余放 rich_text）
  - Familiar（select）：`familiar|unfamiliar|neutral`
  - Created (date)：`createdAt`
  - Extra (rich_text/json)：可保存扩展，如源文/章节等（可选）
- **DB_Highlights**（与 Sentences 关联）
  - Sentence（relation → DB_Sentences）：关联主句（用于聚合）
  - RangeStart（number）：`start`
  - RangeEnd（number）：`end`
  - Color（select）：`yellow|pink|blue|green`
  - Note（rich_text）：`note`
- **DB_Prefs**（可选；也可放在 AppConfig 或 Notion 单页属性）
  - Theme（select）
  - FontSize（number）
  - LineHeight（number）
  - ParagraphSpacing（number）

> 若暂不建独立的 DB_Highlights，也可把高亮序列化为 JSON 写入 DB_Sentences 的一个 rich_text/plain_text 属性，但不利于 Notion 端检索与冲突级别的 Diff 展示。考虑同步队列需要"Diff 与错误定位"能力，推荐高亮独立 DB 并用 relation 连接（方便 DetailsPane 展示差异）【】。

六、同步流程（落地要点）

- **写入策略**：先本地落盘 + 入队（`pending`），后台调用 Notion API 成功后回写 `synced` 与 `updatedAtRemote`（保存远端 `last_edited_time`）；失败则写入 `failed/lastError` 【】【】。
- **拉取策略**：启动用本地缓存渲染 → 后台增量拉取（按 `updatedAtRemote` 游标/Notion `last_edited_time`）并合并【】。
- **冲突处理**：默认 **server_wins**；若本地未同步而远端已改 → 入队 `skipped` 并加"需要处理"标签，或改为 `prompt_user`（设置可切换策略）【】。

- **并发与退避**：并发 3，指数退避+抖动，处理 401/429/5xx/网络错误；401 先刷新再重试一次【】【】。
- **顺序与原子性**：同一实体变更折叠；`delete` 吞并未同步的 `create/update`（避免幽灵对象）【】。
- **自动重试**：网络恢复/令牌刷新后自动拉起 pending/可重试项【】。
- **可视化对齐**：队列项需能展示错误码、重试次数、请求回放片段与本地/远端 ID 对照【】【】。

七、迁移与版本化（建议）

- **Isar schemaVersion**：每次字段调整都增加版本；提供 `fromJson` 兼容旧字段（例如从“本地-only 高亮 JSON”迁到“独立 DB_Highlights”）。
- **软删除**：优先写 `deletedAt`，同步完成后 Notion 端删除（或打 Archive 标记），再异步清理本地物理删除，避免丢数据【】。
- **变更折叠器**：保存到队列前按 `entityType+entityLocalKey` 折叠（只保留最终差异）；列表渲染与统计对齐队列页的 SummaryBar【】。

八、开发对接（最小落地清单）

- **设置页**：Token 输入/测试，DB 链接解析与绑定（支持直接 32 位 database_id）【】【】。
- **列表/详情**：本地渲染与编辑，保存即入队，进入“同步队列视图”观察状态与重试【】。
- **安全**：Token 仅保存在本机安全区（Keystore），不可硬编码【】。

如果你同意这套设计，我可以在下一步直接把 **Isar 的 Dart 模型文件**（含 `@Index`、`@Name`、`@Enumerated` 注解和 `fromNotion()/toNotion()` 适配器雏形）与 **同步调度器接口** 打包给你，用于 Flutter 工程最小运行骨架（与 PRD 的执行表一致：W2/W3 的本地表结构与同步适配层）【】。

notion_csv_templates.zip

isar_notion_sync_starter.zip