在資料庫設計中，當需要儲存「不確定類型」的欄位時，常見的應用場景與解決方案如下：

---

1. 動態表結構需求

應用場景

- **多類型屬性表**：例如電商平台中，不同商品的規格差異極大（如手機有螢幕尺寸，書籍有頁數）。
- **表單/問卷系統**：使用者自定義欄位類型（文字、數字、日期等）。
- **日誌記錄**：需靈活儲存不同事件的異質資料（如錯誤訊息、數值指標、JSON 資料）。

解決方案

- **使用 `sql_variant` 型別**（SQL Server 專用）：

```
CREATE TABLE DynamicAttributes (
    RecordID INT,
    AttributeName NVARCHAR(50),
    AttributeValue SQL_VARIANT  -- 儲存不同類型值
);
```

- **JSON/XML 欄位**：

```
CREATE TABLE EventLogs (
    LogID INT PRIMARY KEY,
    LogData NVARCHAR(MAX)  -- 儲存 JSON/XML 格式資料
);
```

---

2. 跨平台資料整合

應用場景

- **ETL 暫存區**：需暫存來源類型不一致的原始資料（如從 CSV 匯入時部分欄位可能為字串或數字）。
- **API 資料接收**：第三方 API 回傳的欄位類型可能變化（如數值有時以字串格式傳送）。

解決方案

- **寬表設計**：

```
CREATE TABLE StagingData (
    ID INT,
    Value1 NVARCHAR(255),  -- 預留字串空間
    Value2 FLOAT,           -- 預留數值空間
    Value3 DATETIME         -- 預留日期空間
);
```

- **VARCHAR 統一儲存**：

```
CREATE TABLE RawAPIResponses (
    ResponseID INT,
    Content NVARCHAR(MAX)  -- 統一儲存原始字串，後續解析
);
```

---

3. 使用者自定義欄位

應用場景

- **CRM 系統**：客戶可自訂聯絡資訊欄位（如電話、地址、自訂標籤）。
- **低程式碼平台**：允許使用者拖曳建立不同類型的表單欄位。

解決方案

- **EAV 模型**（Entity-Attribute-Value）：

```
CREATE TABLE CustomFields (
    EntityID INT,
    FieldName NVARCHAR(50),
    FieldType NVARCHAR(20),  -- 記錄類型（如 'INT', 'NVARCHAR'）
    FieldValue NVARCHAR(255) -- 統一儲存為字串
);
```

- **混合型別表**：

```
CREATE TABLE UserDefinedData (
    DataID INT,
    StringValue NVARCHAR(255),
    IntValue INT,
    DateValue DATETIME,
    -- 根據實際需求擴充其他類型欄位
);
```

技術選擇對比

方案	優點	缺點	適用場景
`sql_variant`	原生支援多類型，保留原始型別資訊	SQL Server 專用，運算需手動轉換	單一 SQL Server 環境
JSON/XML	跨平台，結構化資料易解析	查詢效能較差，需額外解析	需儲存半結構化資料
EAV 模型	高度彈性，易擴充	查詢複雜，效能差，違反正規化原則	使用者自定義欄位系統
寬表設計	查詢簡單，無需轉換	儲存空間浪費，擴充需修改表結構	欄位類型有限且可預測

實務建議

1. **優先使用 JSON/XML**：若需跨資料庫平台，選擇標準化格式儲存異質資料。
2. **限制 `sql_variant` 使用範圍**：僅用於臨時表或 ETL 過程，避免核心業務表。
3. **搭配應用層驗證**：在寫入資料庫前，於應用層檢查類型合法性（如防止字串誤存為數字）。