日期資料型別時關鍵事項：

一、優先使用現代日期型別

datetime2 取代舊型別：

- 取代 `datetime` 和 `smalldatetime`，支援更廣的日期範圍（0001-01-01 至 9999-12-31）與更高精度（最高 100 奈秒）[1][2][6]。
- 儲存空間彈性（6-8 bytes），依精度需求調整 `datetime2(n)` 參數（例如 `datetime2(3)` 為毫秒精度）[1][4]。

date 與 **time** 分離使用：

- 僅需日期時使用 `date`（3 bytes），僅需時間時用 `time`（3-5 bytes），避免 `datetime` 的冗餘儲存[1][2][4]。

datetimeoffset 處理時區：

- 儲存時區位移（如 `+08:00`），適用於跨時區系統，但儲存成本較高（8-10 bytes）[1][6]。

二、避免已棄用型別與功能

- **棄用型別**：`smalldatetime`（精度僅至分鐘）和 `timestamp`（改用 `rowversion`）[1][3]。
- **混合型別比較**：`datetime` 與 `datetime2` 比較時，SQL Server 2022 強制轉換為 `datetime2`，可能影響效能與結果精度，需統一型別以避免隱性轉換[6]。

三、相容性與升級注意事項

1. **資料庫相容性層級**：
   - 升級至相容性層級 160（SQL Server 2022）時，驗證日期運算與轉換邏輯是否受影響[6]。
2. **隱含轉換風險**：
   - 使用 `CONVERT()` 或 `CAST()` 明確轉換型別，例如將 `GETDATE()`（回傳 `datetime`）轉為 `datetime2` 以匹配其他欄位[1][6]。

四、效能與儲存優化

- **索引設計**：高精度型別（如 `datetime2(7)`）作為索引鍵會增加索引大小，評估是否需降低精度（例如改用 `datetime2(3)`）[2][4]。
- **儲存節省**：
  - 使用 `date` 而非 `datetime2` 可節省 3-5 bytes/筆[2]。
  - 避免 `datetimeoffset` 用於不需時區的場景，減少 8-10 bytes/筆的開銷[1]。

五、函數使用與時區處理

- **統一時間函數**：
  - `SYSDATETIME()` 回傳 `datetime2`，精度優於 `GETDATE()`（回傳 `datetime`）[1]。
  - 使用 `AT TIME ZONE` 轉換時區，取代自行計算位移[6]。
- **時區一致性**：
  - 儲存 UTC 時間（`GETUTCDATE()` 或 `SYSDATETIMEUTCOFFSET()`）以簡化跨時區查詢[1][6]。

六、驗證與測試重點

1. **遷移測試**：
   - 從舊型別（如 `datetime`）轉換至 `datetime2` 時，驗證歷史資料的精度與範圍是否相容[2][6]。
2. **邊界值檢查**：
   - 確認應用程式邏輯是否處理 `datetime2` 的更大範圍（如公元前日期）[1][4]。

透過選擇適當型別、避免棄用功能並優化儲存與查詢，可有效提升 SQL Server 2022 日期資料處理的效能與可靠性。