RDBMS 資料庫設計建議與開發準則

適用聲明

- 本文件（以下簡稱 開發準則）適用於 104corp DBA 團隊所管理的關聯式資料庫管理系統（RDBMS）。
- 適用範疇包含（且不限於）：
  - MariaDB / MySQL / PostgreSQL / AWS Aurora MySQL 等主流 RDBMS 載體。
  - 其他具 RDBMS 資料載體解決方案，請持續參考 104 Technology Standards (https://104cloud.sharepoint.com/:x:/s/EA/EQdML4ub_DZDpkICjMWi1kYBjWWWzuMkrSiu746lh55r8A?e=CeMAEv) & 104 Technology Radar (https://radar.thoughtworks.com/?documentId=https%3A%2F%2Fdocs.google.com%2Fspreadsheets%2Fd%2F1IMGTRbV_V_c9QErkgp4JK4iqZiav_WS5COByLtISA90%2F)
- 註記：
  - 本文件以關聯式資料庫（RDBMS）為「關鍵持久化資料」之主要儲存載體，強調事務性、資料一致性與耐久性（ACID），適用於需長期保存之產品核心資料。
  - 本文件為指引性建議，提供之建議 不具溯及力 ，適用於新開發、重構與需求調整情境。
  - 產品端可依據業務需求與 DBA 討論適度調整；經討論後之調整建議記錄於產品技術文件與架構設計文件。

準則調整與維護辦法

- 本文件由 DBA 團隊負責維護與版本控管。
- 特殊需求應提前與 DBA / 架構團隊評估，並納入架構設計文件。
- 每季檢視及更新快取策略，並應搭配 DBA Quarterly Routine Tasks 季度週期作業 (https://104corp.atlassian.net/browse/ITDBA-3379) 檢討是否存在「快取即資料庫」的錯誤使用並提出修正。

溝通管道：若產品端有相關建議變更提案，請直接聯繫負責 DBA（E-mail 或 Teams），DBA 會在季度例會

修訂歷史

版本	日期	修訂者	審核者	變更摘要 & 影響章節	關聯 Ticket
1.0	2025/10/07	DBA Team	DBA Team	初版發布	ITDBA-3379 (https://104corp.atlassian.net/browse/ITDB 3379)

標準化設計建議與開發準則

說明：本文件針對 RDBMS 操作設計提供標準化建議、正確 / 不正確範例與對產品應用的影響分析，方便開發/運維落地執行與稽核。

Global

- Charset：
  - [MUST] 使用 utf8mb4 作為唯一字符設定。

    ```
    CHARACTER SET != utf8mb4; -> 無法安全儲存 4-byte 字元（emoji）會亂碼或截斷。
    ```

  - [SHOULD] 選用 utf8mb4_unicode_ci 或 utf8mb4_0900_ai_ci 作為 collation。

    ```
    若環境為 MySQL 8.0+：優先使用 utf8mb4_0900_ai_ci（Unicode 9.0、排序更正確、效能
    若需跨 MariaDB / 舊 MySQL 兼容：選 utf8mb4_unicode_ci（較廣泛支援）。
    ```

  - [MUST NOT] 混用不同字符集/排序規則。

    ```
    混用不同 charset/collation（join/cmp）
    JOIN/比較兩表 charset 不一致，會出現 collation 錯誤或隱性轉換效能問題。
    EX: **"Illegal mix of collations"**。
    ```

- 正規化與反正規化：
  - [MUST] 設計符合正規化。

    正規化優點：避免資料重複、簡化一致性、較小更新成本。

    - [MAY] 必要時適度反正規化並記錄理由與影響。

      為了效能或簡化查詢，反正規化是優化手段，不是設計預設。

- 分庫分表：
    - [MAY] 無統一強制標準；產品團隊需求發生時，請與 DBA 依業務需求共同制定方案。
- 避免 TEXT/BLOB 大欄位設計：
    - [SHOULD NOT] 直接使用 TEXT/BLOB 存大量資料。
        - 不要把媒體/大文件放在核心表格裡，會破壞查詢效能與備份 SLA。

      大欄位會造成 `IO/備份/複製/binlog` 膨脹、查詢延遲、鎖競爭與恢復時間暴增。

    - [SHOULD] 對結構化大型資料考慮 JSON（MySQL 5.7+）或 NoSQL。
        - JSON 優勢：可用 JSON_EXTRACT、generated column 建索引、保有部分 SQL 查詢能力。

      大檔案優先：外部物件儲存 `+ DB metadata (storage_uri, checksum, size)`。

- Schema 與註解：
    - [MUST] 表加 comment。

      快速理解表用途、`Owner`、保留/刪除策略。

    - [SHOULD] 欄位加 comment 幫助維護。

      欄位註解說明語意、單位、`enum` 值、預期長度與約束，減少錯誤使用與溝通成本。

- 主鍵設計：
    - [MUST NOT] 主鍵綁定業務欄位。

      業務邏輯可能會改變（例如 `email`、手機號碼、身分證字號可能因更正或規範變更而異動），
      若被設為 `Primary Key` 將極大增加 `schema` 變更與維護成本。

    - [SHOULD] 主鍵需穩定、唯一、簡潔且不可變。

      – 建議使用應用端生成之 `Surrogate Key`（例如時序型 `UUID[v7]`、`ULID`、`Snowflake` 或序
      – 且不受業務規則影響，同時索引結構更簡潔，提升 `join/查詢效率及資料一致性維護。`

- NULL 值處理：
    - [SHOULD] 理解 NULL 語義（未知/不適用）。

NULL 表示「值未知或不適用」，非空字串/0 的別名。設計欄位時先判斷語意再決定是否允許 NUL
範例說明：birth_date 可以為 NULL（未知生日）；age 不該 NULL（若不能計算用 default

- [SHOULD NOT] 對頻繁查詢欄位允許 NULL。

  NULL 會導致額外的 IS NULL 判斷、索引使用率降低、統計不一致及執行計畫差異。
  對高頻查詢欄位應使用 NOT NULL 並提供合理 default 或 refactor schema。

- [MAY] 在特定情況允許 NULL（如搭配 unique key）。

  – MySQL 的 UNIQUE index 允許多個 DEFAULT NULL 但不允許多個 DEFAULT ""。

- [SHOULD] 確保應用層傳入值與欄位型態一致。

  API/服務端驗證、ORM mapping 與 DB 嚴格模式應保持一致。在啟用 SQL strict 模式，避

- 時間欄位：
  - [SHOULD] 對 create/update 時間欄位可採資料庫自動維護（DEFAULT CURRENT_TIMESTAMP / ON UPDATE）或由應用端填值，且應依欄位需求選擇 NOT NULL 或 NULLABLE 策略。
  - [MAY] 在必要時使用 TIMESTAMP（注意範圍限制）
  - [SHOULD] 優先使用 DATETIME

    DATETIME 範圍寬（1000–9999），TIMESTAMP 範圍有限（1970–2038）。
    TIMESTAMP 會依 session.time_zone 自動轉換，容易造成不可預期值（若未統一時區）。

  - [MUST] 明訂 (跨時區 / 日光節約時間) 策略。

    應用端顯示時依使用者時區轉換，避免 DST 導致時間錯置/重疊導致商務邏輯錯誤。

查（Read）

- 索引設計：
  - [MUST] 為每表之 主鍵 與 外鍵 （即使不啟用 FK 約束）建立索引。

    主鍵天然具唯一性，索引可確保檢索效能。
    外鍵即使未啟用約束仍常用於 JOIN 操作，建立索引可避免掃描整表並提升一致性檢查效率。

  - [SHOULD] 建立覆蓋常用查詢條件之索引。

須分析業務查詢模式與執行計畫，針對 WHERE、JOIN 及 ORDER BY 高頻欄位建立複合或覆蓋索
以減少 I/O 與避免全表掃描。避免過度索引造成維護負擔與 DML 效能下降。

- [MUST] 遵循最左前綴原則。

  複合索引必須依照查詢條件的最左欄位開始比對，否則將無法有效利用索引。
  例如 (col_a, col_b, col_c) 索引，僅查詢 col_b、col_c 時無法套用。

- [SHOULD] 高選擇性欄位放前。

  索引欄位排列建議將資料分布差異大、過濾效果佳（高選擇性）的欄位置於前面，
  以有效減少 DataSet，提高索引利用率與查詢效能。
  高選擇性: 身分證字號、手機號碼 等唯一資料屬性
  低選擇性: 性別、Boolean、狀態碼

- [SHOULD] 為 ORDER BY / GROUP BY 欄位考慮索引。

  若排序或彙總欄位頻繁出現，適當建立索引可減少額外排序資源消耗。

  ```
  EX: CREATE INDEX idx_table_colA_colB ON my_table (col_a, col_b);

  -- 查詢範例：透過索引支援 ORDER BY
  SELECT * FROM my_table WHERE col_a > 100
  ORDER BY col_a, col_b;

  -- 查詢範例：透過索引支援 GROUP BY
  SELECT col_a, col_b, COUNT(*) FROM my_table
  GROUP BY col_a, col_b;
  ```

- [SHOULD] 定期檢視並移除未使用或低效索引。

  長期累積的無用索引可能導致 INSERT/UPDATE/DELETE 作業時，均會對索引樹增加操作行為導

- [SHOULD NOT] 對索引欄位使用函數（導致索引失效，例如 DATE(col) ）。

  若查詢條件對索引欄位使用函數（如 `DATE(col)`、`UPPER(col)`），
  資料庫無法直接利用該索引，會導致全表掃描，效能下降。

- [SHOULD] 使用明確起訖範圍查詢（BETWEEN/區間）。

查詢若能限制清楚的起點與終點，能讓資料庫更有效利用索引做範圍掃描，避免不必要的全表掃描。
EX:

```
-- 不建議：使用函數或模糊條件
SELECT * FROM logs WHERE DATE(log_time) = '2025-09-15';

-- 建議：使用明確範圍
SELECT * FROM logs
WHERE log_time >= '202X-YY-ZZ 00:00:00'
  AND log_time <  '202X-YY-ZZ 00:00:00';
```

- 查詢規範：
  - [MUST NOT] 生產環境使用 `SELECT *` 。

    使用 `SELECT *` 會導致：多撈取非必要欄位，加大網路傳輸成本，增加資料庫 I/O 負擔與記
    EX:

    ```
    -- 不建議
    SELECT * FROM customers WHERE id = 1001;

    -- 建議
    SELECT id, name, email FROM customers WHERE id = 1001;
    ```

  - [SHOULD] 明確列出欄位並加 `LIMIT` 。

    建議加上 `LIMIT`，限制回傳筆數，降低記憶體壓力並避免應用端誤用全表查詢。
    EX:

    ```
    -- 不建議
    SELECT * FROM orders;

    -- 建議
    SELECT order_id, customer_id, total_amount FROM orders ORDER BY created_
    LIMIT 100;
    ```

  - [SHOULD NOT] 在大表使用前綴模糊匹配 `LIKE '%value%'` 。

    使用前綴模糊匹配 (`%value%`) 會導致索引失效，
    查詢需進行全表掃描 (Full Table Scan)，在大表上效能極差。

  - [MAY] 若需全文檢索，改用專門搜尋引擎。
- 檢視（View）：
  - [MUST] 命名需明確且描述用途。

View 命名應符合資料庫物件命名規範，避免簡略或模糊不清。
建議包含業務領域與功能描述，方便維運與開發理解。
EX:

```
-- 不建議
CREATE VIEW v1 AS ...

-- 建議
CREATE VIEW sales_monthly_summary AS ...
CREATE VIEW customer_active_orders AS ...
```

- [MUST NOT] 使用 `SELECT *` 。

明確定義 View 中所需欄位，保持欄位清單穩定。
僅保留必要資料，減少無關欄位載入。
EX:

```
-- 不建議
CREATE VIEW v_order_summary AS SELECT * FROM orders;

-- 建議
CREATE VIEW v_order_summary AS SELECT order_id, customer_id, order_date,
```

- [MUST NOT] 多層巢狀。

多層 View 巢狀會導致查詢計劃複雜，效能下降。
僅允許 View 基於單層資料表或已優化的基礎 View。
EX:

```
-- 不建議
CREATE VIEW v_customer AS
SELECT * FROM v_customer_base;  -- v_customer_base 又依賴 v_user_info

-- 建議
CREATE VIEW v_customer AS
SELECT c.customer_id, c.name, u.email
FROM customers c
JOIN users u ON c.user_id = u.user_id;
```

- [SHOULD NOT] 避免複雜子查詢於 View。

```
-- 不建議
```

複雜子查詢（尤其含多重 JOIN、相關子查詢或聚合）會讓 View 難以維護與調試。
對查詢最佳化器不友善，可能產生非預期的執行計劃，導致效能低落。

```
-- 建議
```

保持 View 輸出精簡，僅作為欄位或表格對應的封裝層。

- [MUST NOT] View 內不做動態過濾；在外層條件會造成全表掃描

```
-- 不建議
View 並不支援參數化，若在應用程式上以「帶條件」方式使用 View，RDBMS 往往會先完整展開

-- 建議
僅在 View 內保留固定的業務邏輯，不做動態過濾。
```

增（Create / Insert）

- 主鍵與 ID 生成：
  - [SHOULD NOT] 使用 AUTO_INCREMENT（建議應用層生成全局唯一 ID）；
  - [SHOULD] 應用層生成全域唯一識別碼（例如：時序型 UUID[v7]、ULID、Snowflake）作為 Primary Key。

    ```
    -- 不建議
    AUTO_INCREMENT 屬於單節點序號生成方式，
    在分散式 / 多節點架構下會導致 ID 重疊、熱點集中，難以橫向擴展。

    -- 建議
    去中心化，避免單點生成瓶頸。
    可在多服務、多資料中心中保證唯一性。
    部分演算法（ULID / Snowflake）具排序性，可減緩索引分裂。
    ```

  - [SHOULD] 將 `UUID` 儲為 `BINARY(16)` 而非 `VARCHAR(36)`。

    ```
    BINARY(16) 僅需 16 Bytes，
    VARCHAR(36) 為字串，存儲空間與索引成本更高，效能較差。
    ```

- 欄位預設：
  - 文字欄位：
    - [SHOULD] `DEFAULT ''` 而非 `NULL`。

      ```
      -- 建議
      避免 NULL 判斷造成 SQL 較複雜（需 IS NULL）。
      提高索引利用率，因 NULL 會造成統計分佈與索引效率下降。
      ```

  - 數字欄位：
    - [SHOULD] `DEFAULT 0` 或 `DEFAULT -1`（特殊值須記錄）。

      ```
      建立明確初始值，避免 NULL 導致運算/比較異常；
      若需代表特殊狀態（如 -1 = 未初始化），必須在 Schema 文件中清楚註記其語意。
      ```

    - [MUST] 為數字欄位定義明確業務含義。

每個數字欄位需對應明確業務邏輯，避免通用數字欄位導致誤用。

- [SHOULD] 金額使用 DECIMAL 而非 FLOAT/DOUBLE。

  DECIMAL 保證小數精確度，適合金額/會計應用；

- [MUST] 為不會為負值欄位指定 UNSIGNED（如適用）。

  明確表達值域，避免誤存負數；
  EX: 計數器、年齡、庫存、積分 等應使用 UNSIGNED。

- 型別一致性：
  - [MUST] 確保應用層傳入值型別與 DB 欄位一致，避免數字存字串。
    - 不可用 VARCHAR 儲存數字，以免：
      - 無法利用索引進行數值比較與排序；
      - 容易出現隱性轉型，影響效能與正確性；
      - 難以約束值域範圍。

      ```
      EX:
      -- 不建議
      -- 使用 VARCHAR 儲存年齡
      CREATE TABLE user_varchar_age (
          id INT PRIMARY KEY AUTO_INCREMENT,
          age VARCHAR(3) NOT NULL
      );

      -- 可以插入無效值，難以約束範圍
      INSERT INTO user_varchar_age (age) VALUES ('abc');
      INSERT INTO user_varchar_age (age) VALUES ('-5');
      INSERT INTO user_varchar_age (age) VALUES ('200');

      -- 建議
      -- 使用 INT 並限制值域
      CREATE TABLE user_int_age (
          id INT PRIMARY KEY AUTO_INCREMENT,
          age TINYINT UNSIGNED NOT NULL CHECK (age BETWEEN 0 AND 120)
      );

      -- 若插入無效值會直接失敗
      INSERT INTO user_int_age (age) VALUES (-5);    -- 失敗
      INSERT INTO user_int_age (age) VALUES (200);   -- 失敗
      ```

- 大量寫入：
  - [SHOULD] 避免大事務。
    - 單一事務若操作筆數過大，會造成：
      - 鎖範圍過廣，影響其他交易讀寫；

- Undo/Redo Log 暴增，降低效能；
- 一旦失敗回滾，成本極高。

```
-- 建議
單次事務操作控制在可容忍範圍內。
```

- [SHOULD] 分批或採異步寫入（批次處理資料）。

```
-- 建議
將大量資料寫入分批處理，例如每批 1,000 筆。
```

- [MAY] 依業務允許採用批次/批量插入優化。

```
-- 建議
小筆 INSERT 頻繁提交成本高，
可透過批次 INSERT 減少網路往返與事務開銷。
EX:
INSERT INTO orders (id, user_id, amount) VALUES
(1, 1001, 99.9),
(2, 1002, 120.5),
(3, 1003, 45.0);
```

刪（Delete）

- 安全機制：
  - [MUST NOT] 執行無 WHERE 的 DELETE（同時適用於 SELECT/UPDATE）。

```
-- 不建議
禁止如下語句：
DELETE FROM users;
UPDATE users SET status = 'inactive';

-- 建議
需強制條件限制：
DELETE FROM users WHERE id = 1001;
```

  - [SHOULD] 刪除操作限制每次影響行數並分批執行

大量資料筆數作業宜分批刪除（例如每次刪 1,000）；避免長事務與大範圍鎖表。

  - [SHOULD] 刪除條件需有索引以避免全表掃描。

```
-- 建議
刪除語句必需與索引條件搭配，避免 full table scan。
若作業資料量大，可考慮使用分區表（時間/範圍）。
```

- 資料復原策略：
  - [MUST] 刪除重要資料前應備份或使用軟刪除（視業務需求）。

    ```
    -- 建議
    批次刪除前：於欄位 is_deleted / deleted_at，改採軟刪除機制。
    或保留臨時歸檔表，若誤刪可從該表回補。
    ```

改（Update / Schema 變更）

- Schema 變更管理：
  - [SHOULD] 如果允許，使用 Migration（遷移） 工具管理結構變更。
  - [MUST] 變更前需備份、風險評估與回滾計畫。
  - [MUST] DBA 審核後執行變更。
- 外鍵（Foreign Key）約束：
  - [SHOULD NOT] 在生產環境啟用 FK constraint（以避免效能影響）。
    - FK constraint 雖可以強化一致性，但會增加營運 Online Schema Change 複雜度。

    ```
    -- 建議
    傾向由應用邏輯與巡檢機制替代 FK。
    若未妥善設計，將導致髒資料產生，衍伸資料一致性維護困難問題。
    ```

- 索引管理：
  - [SHOULD] 新增/移除索引前評估對寫入/查詢的影響，並定期清理未使用索引。

    ```
    -- 建議
    在新索引建立前，應透過 EXPLAIN、慢查詢日誌 分析查詢效益；

    -- 不建議
    直接新增索引來嘗試「解決所有慢查詢」，
    會造成 INSERT/UPDATE/DELETE 成本上昇，甚至引入多餘冗餘索引。
    ```

- Stored Procedure / Function：
  - [MUST] 命名慣例統一並描述用途。

    ```
    -- 不建議
    使用模糊或簡略命名，例如：
    proc1(), myfunc()
    ```

  - [MUST] 避免邏輯過於複雜或跨表大量操作。

```
-- 不建議
Stored Procedure 應保持單一職責，避免實作過多業務邏輯或跨數百萬筆大表 Join，以免難
```

- [SHOULD] 加入適度 exception handler。

```
-- 建議
使用 DECLARE ... HANDLER 捕捉錯誤並回傳錯誤碼，降低交易中斷風險，利於應用端 Debug
```