Khóa SQL — Tài liệu học viên

Đi nhanh tới

Hướng dẫn sử dụng bài giảng

Buổi 1 | Buổi 2 | Buổi 3 | Buổi 4 | Buổi 5 | Buổi 6 | Buổi 7 | Buổi 8 | Buổi 9 | Buổi 10

- Phần I — Hướng dẫn sử dụng bài giảng 6
- Phần II — Nội dung bài giảng 8
- Buổi 1 — Giới thiệu SQL và cấu trúc câu lệnh SELECT cơ bản 9
  - Mục tiêu buổi học 9
  - Bản đồ buổi học 10
  - 1. SQL là gì? Tại sao cần học SQL? 10
  - 2. Bộ dữ liệu mẫu của khóa học 12
  - 3. Công cụ và cách chạy bài trên Codelab 15
  - 4. Cấu trúc cơ bản của câu lệnh SELECT 16
  - 5. Mệnh đề FROM và WHERE 18
  - 6. Mệnh đề ORDER BY 25
  - 7. Áp dụng: từ yêu cầu nghiệp vụ đến câu lệnh 26
  - Sai lầm thường gặp 27
  - Tóm tắt 28
  - Gợi ý mở rộng 28
  - Bài tập thực hành của buổi 29
  - Thực hành trên Windows: cài đặt và làm quen công cụ 29
- Buổi 2 — Sắp xếp dữ liệu và các hàm thống kê cơ bản 33
  - Mục tiêu buổi học 33
  - Bản đồ buổi học 34
  - 1. Ôn tập và mở rộng ORDER BY 34
  - 2. Các hàm thống kê tổng hợp cơ bản 35
  - 3. Mệnh đề giới hạn số dòng: TOP, LIMIT và OFFSET 38
  - 4. Mệnh đề GROUP BY 41
  - 5. Mệnh đề HAVING 45
  - 6. Các hàm tổng hợp nâng cao 46
  - 7. Áp dụng: từ câu hỏi nghiệp vụ đến báo cáo 50
  - Sai lầm thường gặp 51
  - Tóm tắt 51
  - Gợi ý mở rộng 52
  - Bài tập thực hành của buổi 52
  - Thực hành trên Windows: xem, đếm và xuất kết quả báo cáo 52
- Buổi 3 — Window Functions (hàm cửa sổ) 54
  - Mục tiêu buổi học 54

- Bản đồ buổi học 54
- 1. Tổng quan Window Functions 55
- 2. Window Aggregate Functions: SUM / AVG / MIN / MAX / COUNT ... OVER 59
- 3. Ranking Functions: ROW_NUMBER, RANK, DENSE_RANK 62
- 4. LAG và LEAD — phân tích biến động 65
- 5. Window Frame và Moving Average 68
- 6. Tổng kết và Q&A 73
- Sai lầm thường gặp 74
- Bài tập thực hành của buổi 74
- Thực hành trên Windows: so sánh GROUP BY với Window Function 75

- Buổi 4 — Kiểu dữ liệu và hàm xử lý dữ liệu 77
  - Mục tiêu buổi học 77
  - Bản đồ buổi học 77
  - 1. Các kiểu dữ liệu phổ biến 78
  - 2. Hàm xử lý chuỗi 83
  - 3. Hàm xử lý ngày giờ 88
  - 4. Hàm toán học 94
  - 5. Hàm điều kiện 97
  - 6. Tổng kết 102
  - Sai lầm thường gặp 102
  - Bài tập thực hành của buổi 103
  - Thực hành trên Windows: kiểu dữ liệu, ngôn ngữ và tiếng Việt 104
- Buổi 5 — CTE, TEMP TABLE và Subquery (truy vấn con) 106
  - Mục tiêu buổi học 106
  - Bản đồ buổi học 106
  - 1. Subquery là gì? Tại sao cần? 107
  - 2. Subquery trong WHERE 109
  - 3. Subquery trong SELECT và FROM 113
  - 4. Correlated Subquery (subquery tương quan) 115
  - 5. CTE — Common Table Expression 116
  - 6. TEMP TABLE — Bảng tạm 119
  - 7. Thực hành tổng hợp: Subquery → CTE → TEMP TABLE 122
  - Sai lầm thường gặp 123
  - Tóm tắt 124
  - Gợi ý mở rộng 124
  - Bài tập thực hành của buổi 124
  - Thực hành trên Windows: bảng tạm, CTE và đo thời gian chạy 125
- Buổi 6 — Thay đổi dữ liệu: INSERT, UPDATE, DELETE 127
  - Mục tiêu buổi học 127
  - Bản đồ buổi học 127
  - 1. Lệnh INSERT — thêm dữ liệu 128
  - 2. Lệnh UPDATE — sửa dữ liệu 132
  - 3. Lệnh DELETE — xóa dữ liệu 135
  - 4. Transaction (giao dịch) 139
  - 5. Best practices và bảo mật 144
  - Sai lầm thường gặp 146

- Tóm tắt 146
- Gợi ý mở rộng 147
- Bài tập thực hành của buổi 147
- Thực hành trên Windows: sửa dữ liệu an toàn và dùng transaction 147

- Buổi 7 — Mối quan hệ giữa bảng & Normal Form 149
  - Mục tiêu buổi học 149
  - Bản đồ buổi học 150
  - 1. Khái niệm mối quan hệ (Relationships) 150
  - 2. Khóa (Key) trong Database 153
  - 3. Normalization (Chuẩn hóa) 159
  - 4. ER Diagram (Entity-Relationship) 164
  - Sai lầm thường gặp 167
  - Tóm tắt 168
  - Bài tập thực hành của buổi 168
  - Gợi ý mở rộng 169
  - Thực hành trên Windows: khóa, sơ đồ ER và tạo bảng bằng giao diện 169
- Buổi 8 — JOIN: Kết hợp bảng 171
  - Mục tiêu buổi học 171
  - Bản đồ buổi học 172
  - Tổng quan các loại JOIN 172
  - 1. INNER JOIN 172
  - 2. LEFT JOIN (LEFT OUTER JOIN) 176
  - 3. RIGHT JOIN (RIGHT OUTER JOIN) 178
  - 4. FULL JOIN (FULL OUTER JOIN) 179
  - 5. CROSS JOIN 181
  - 6. Self Join 182
  - 7. Phân tích bằng JOIN 184
  - Sai lầm thường gặp 186
  - Tóm tắt 187
  - Bài tập thực hành của buổi 187
  - Gợi ý mở rộng 187
  - Thực hành trên Windows: dựng JOIN bằng giao diện và đọc kế hoạch thực thi 188
- Buổi 9 — Stored Procedure & Trigger (cơ bản) 190
  - Mục tiêu buổi học 190
  - Bản đồ buổi học 190
  - 1. Stored Procedure là gì? 191
  - 2. Viết Stored Procedure 194
  - 3. Gọi Stored Procedure 200
  - 4. Trigger là gì? 203
  - 5. Viết Trigger 204
  - 6. Best Practices 209
  - Sai lầm thường gặp 210
  - Tóm tắt 211
  - Gợi ý mở rộng 211
  - Bài tập thực hành của buổi 211
  - Thực hành trên Windows: tạo, chạy và quản lý Procedure, Trigger 212

- Buổi 10 — Index & Transaction + Review toàn bộ khóa 214
  - Mục tiêu buổi học 214
  - Bản đồ buổi học 214
  - 1. Index (chỉ mục) là gì? 215
  - 2. Các loại Index 218
  - 3. Tạo và quản lý Index 220
  - 4. Transaction (giao tác) và toàn vẹn dữ liệu 226
  - 5. Hiệu năng (Performance Tuning) 232
  - 6. Kết nối và trích xuất dữ liệu SQL trực tiếp từ Excel 233
  - 7. Review và hệ thống hóa toàn bộ khóa học 236
  - Sai lầm thường gặp 240
  - Tóm tắt 241
  - Gợi ý mở rộng 241
  - Bài tập thực hành của buổi 241
  - Thực hành trên Windows: đo hiệu năng index, quan sát khóa và nối Excel 242

Gồm: Phần I — hướng dẫn sử dụng bài giảng; Phần II — toàn bộ nội dung 10 buổi.

Phần I — Hướng dẫn sử dụng bài giảng

Về mục lục

Tài liệu hướng dẫn sử dụng bài giảng khóa SQL (10 buổi, 30 giờ) trên Codelab: đọc bài giảng, xem slide, chạy thử ví dụ trên trình duyệt và nộp bài tập chấm tự động.

Địa chỉ: https://codelab.ai.vn/codelab/subjects/sql

1. Bắt đầu

1. Mở địa chỉ trên và bấm **Bắt đầu học — Buổi 1**, hoặc chọn một buổi trong **Lộ trình 10 buổi**.
2. **Đăng nhập** để lưu tiến độ và nộp bài (đăng nhập một lần của đơn vị nếu có, Google hoặc email). Bạn vẫn đọc bài và bấm **Chạy thử** được khi chưa đăng nhập.
3. Công cụ chính của khóa là **SQL Server + SSMS**. Phần **Thực hành trên Windows** ở cuối Buổi 1 hướng dẫn từng bước cài đặt; làm trước buổi học đầu tiên nếu có thể.

2. Cách sử dụng tài liệu bài giảng

Mỗi buổi học là một trang. Cách học mỗi buổi:

1. **Đọc mục tiêu và bản đồ buổi học** để biết sẽ học gì và thời gian dành cho từng phần.
2. **Đọc bài giảng theo thứ tự các mục** (dùng mục lục cuộn bên cạnh để nhảy mục). Với mỗi ví dụ SQL, bấm **Chạy thử** để xem kết quả, rồi sửa câu lệnh và chạy lại để hiểu.
3. **Làm ngay khung Thực hành** xuất hiện sau mỗi phần lý thuyết, rồi mới đọc tiếp.
4. **Cuối buổi:** đọc *Sai lầm thường gặp* và *Tóm tắt*, làm nốt các bài còn lại trong danh sách *Bài tập thực hành của buổi*.
5. **Làm lại trên máy của bạn** theo phần *Thực hành trên Windows* ở cuối buổi, để quen với công cụ thật.
6. Cần đọc ngoại tuyến thì bấm 🖨️ **Tải PDF / In**, hoặc dùng bản PDF đầy đủ của khóa.

Các thành phần của một buổi học:

Thành phần	Dùng để
Mục tiêu và bản đồ buổi học	Biết sẽ học gì, thời gian dành cho lý thuyết và thực hành
Bài giảng	Mục lục cuộn bên cạnh, hộp **Lưu ý / Mẹo / Cảnh báo**, sơ đồ dữ liệu minh họa
Ví dụ SQL có nút Chạy thử	Bấm để chạy ngay, xem bảng kết quả; chọn **SQL Server** hoặc **MySQL** nếu khối có cả hai
Khung Thực hành	Bài tập xen kẽ trong bài giảng; bấm vào để mở đề và nộp bài
Sai lầm thường gặp, Tóm tắt, Gợi ý mở rộng	Ôn lại cuối buổi
Thực hành trên Windows	Các thao tác trên SSMS dành cho buổi đó (làm tại máy của bạn)
Bài tập thực hành của buổi	Danh sách đầy đủ bài của buổi

Mỗi buổi có nút 🖥️ **Trình chiếu** (xem slide; phím ←/→ chuyển slide, F toàn màn hình, Esc thoát) và 🖨️ **Tải PDF / In** để lưu bài giảng thành PDF. Nút 🧰 **Công cụ** mở sân chơi SQL, bảng cú pháp, lược đồ và tab **Dữ liệu & tải về**.

3. Làm và nộp bài

1. Mở bài, đọc đề, xem bảng dữ liệu liên quan và ví dụ mẫu.
2. Chọn ngôn ngữ **SQL Server (T-SQL)** (khuyến nghị) hoặc **MySQL**.
3. Viết lời giải (đúng một câu `SELECT` trừ khi đề nói khác), bấm **Chạy thử** để xem kết quả rồi **Nộp bài**.
4. Hệ thống chạy bài của bạn trên **nhiều bộ dữ liệu khác nhau** (có NULL, giá trị biên...) và so khớp **từng dòng** với đáp án: đúng **thứ tự cột**, đúng thứ tự dòng theo `ORDER BY`, không in tên cột.

Quy ước cần nhớ:

- "Hôm nay" của dữ liệu là **2026-09-25**: dùng ngày cố định này, **không dùng** `GETDATE()` / `NOW()` trong lời giải.
- Viết đúng chữ hoa/thường của tên bảng và cột (`Customers`, `MarketPrices`...).
- Đừng viết cứng kết quả theo dữ liệu mẫu (ví dụ `WHERE customer_id IN (2, 4, 5)`): bộ dữ liệu ẩn sẽ làm lời giải sai.
- Bài T-SQL tạo procedure/trigger/view cần dòng `GO` ở đầu script.

4. Tiến độ và hoàn thành khóa

- Trang khóa học hiện **Tiến độ của bạn** (số bài đã đạt trên tổng số).
- **Hoàn thành khóa:** giải được ít nhất **70%** số bài thực hành. Khi đủ điều kiện, bấm cấp **giấy xác nhận** (có mã, kiểm tra được trực tuyến).
- Bài đạt khi nộp đúng toàn bộ bộ dữ liệu; nếu sai, đọc thông báo kết quả, sửa và nộp lại không giới hạn.

5. Tài liệu bổ sung

- **Tài liệu & hướng dẫn cài đặt môi trường:** https://codelab.ai.vn/codelab/courses/sql/tai-lieu (tải bộ dữ liệu mẫu cho SQL Server, MySQL, PostgreSQL, SQLite).

Phần II — Nội dung bài giảng

Về mục lục

Mười buổi học theo thứ tự. Các ví dụ SQL trong bản này không có nút Chạy thử; hãy làm trên trang trực tuyến hoặc trên máy của bạn.

Buổi 1 — Giới thiệu SQL và cấu trúc câu lệnh SELECT cơ bản

Về mục lục

Chào mừng các anh chị đến với buổi đầu tiên của khóa đào tạo SQL. Trong 3 giờ tới chúng ta sẽ đi từ câu hỏi "SQL là gì?" đến việc tự viết được những truy vấn lọc và sắp xếp dữ liệu khách hàng, tài khoản, lệnh giao dịch – đúng loại yêu cầu mà bộ phận IT, phân tích và back-office của một công ty chứng khoán gặp hằng ngày: *"cho tôi danh sách tài khoản số dư lớn"*, *"khách hàng nào chưa có email để gửi thông báo?"*, *"lệnh bán nào khớp gần đây?"*.

Cách học của khóa: **học đến đâu làm đến đó**. Sau mỗi phần lý thuyết ngắn sẽ có khung **Thực hành** ngay tại chỗ; các ví dụ SQL trong bài có nút **Chạy thử** để bạn tự bấm xem kết quả trên bộ dữ liệu mẫu.

Mục tiêu buổi học

Sau buổi này bạn có thể:

- Giải thích SQL là gì, các khái niệm Database, Table, Row, Column và vai trò của SQL trong công ty chứng khoán.
- Đọc hiểu cấu trúc bộ dữ liệu mẫu của khóa học (Customers, Accounts, Securities, MarketPrices, Orders, Trades, Portfolio, CustomerStats, AuditLog) và các quan hệ giữa chúng.
- Viết câu lệnh `SELECT ... FROM ...`, đặt bí danh cột và loại trùng bằng `DISTINCT`.
- Lọc dữ liệu bằng `WHERE` với các toán tử so sánh, `AND` / `OR` / `NOT` (đúng độ ưu tiên), `LIKE`, `IN`, `BETWEEN`, `IS NULL`.
- Sắp xếp kết quả với `ORDER BY` (ASC/DESC, nhiều cột, thứ tự ưu tiên).
- Chạy và nộp bài SQL trên Codelab bằng SQL Server (T-SQL) hoặc MySQL.

Bản đồ buổi học

Phần	Nội dung	Hình thức	Thời lượng
1	SQL là gì, vai trò, các hệ quản trị CSDL	Lý thuyết	12 phút
2–3	Bộ dữ liệu mẫu, công cụ, cách nộp bài trên Codelab	Lý thuyết + demo	13 phút
4	Cấu trúc `SELECT`, bí danh, `DISTINCT` + **bài 1.1**	7 phút lý thuyết + 18 phút thực hành	25 phút
5.1–5.3	`FROM`, `WHERE`, so sánh, `AND` / `OR` / `NOT` + **bài 1.2**	8 phút lý thuyết + 27 phút thực hành	35 phút
5.4–5.5	`LIKE`, `IN`, `BETWEEN`, `NULL` + **bài 1.3**	8 phút lý thuyết + 34 phút thực hành	42 phút
6	`ORDER BY` + **bài 1.4**	8 phút lý thuyết + 36 phút thực hành	44 phút
7–9	Áp dụng nghiệp vụ, sai lầm thường gặp, tóm tắt, chữa bài, Q&A	Thực hành + thảo luận (làm lại bài khó, đổi chéo chữa bài)	9 phút
	Tổng	~56 phút lý thuyết + demo, ~124 phút thực hành (~69%)	**180 phút**

1. SQL là gì? Tại sao cần học SQL?

SQL (Structured Query Language) là ngôn ngữ chuẩn để *hỏi* và *thao tác* dữ liệu trong cơ sở dữ liệu quan hệ. Bạn không cần chỉ cho máy "làm thế nào" (duyệt từng dòng, so sánh từng ô); bạn chỉ mô tả "muốn lấy dữ liệu gì", hệ quản trị tự tìm cách nhanh nhất.

> **Ví dụ nghiệp vụ:** cuối ngày, phòng Quản lý rủi ro cần danh sách tài khoản ký quỹ có số dư dưới ngưỡng. Với Excel, bạn phải mở file, lọc, sao chép thủ công mỗi ngày. Với SQL, một câu lệnh 3 dòng chạy trên dữ liệu thật, luôn mới nhất, và tái sử dụng được mỗi ngày.

1.1. Bốn khái niệm nền tảng

Khái niệm	Ý nghĩa	Ví dụ trong công ty chứng khoán
Database (cơ sở dữ liệu)	Nơi lưu toàn bộ dữ liệu của một hệ thống	CSDL `hnx_securities` của khóa học
Table (bảng)	Một tập hợp dữ liệu cùng chủ đề, giống một trang tính Excel	`Customers` (khách hàng), `Orders` (lệnh)
Row (hàng, bản ghi)	Một đối tượng cụ thể trong bảng	Khách hàng "Nguyen Van An"
Column (cột, trường)	Một thuộc tính của đối tượng, có kiểu dữ liệu cố định	`full_name`, `city`, `birth_date`

Hình dung bảng `Customers` như sau (mỗi **dòng ngang** là một khách hàng, mỗi **cột dọc** là một thuộc tính):

customer_id	full_name	customer_type	city
1	Nguyen Van An	INDIVIDUAL	Ha Noi
2	Tran Thi Binh	INDIVIDUAL	Da Nang
31	Ngan Hang TMCP Dau Tu Viet	INSTITUTION	TP.HCM

1.2. Vai trò của SQL trong công ty chứng khoán

- **Báo cáo**: số lượng tài khoản mở mới trong tháng, tổng giá trị khớp lệnh theo mã, báo cáo gửi Sở giao dịch.
- **Phân tích**: nhóm khách hàng theo thành phố/độ tuổi, mã cổ phiếu được giao dịch nhiều nhất, hiệu quả từng kênh giới thiệu.
- **Kiểm soát rủi ro**: tài khoản margin có số dư thấp, lệnh bất thường về khối lượng, khách hàng bị khóa vẫn phát sinh lệnh.
- **Vận hành/IT**: đối soát dữ liệu giữa các hệ thống, kiểm tra dữ liệu sau khi nâng cấp, sửa dữ liệu lỗi có kiểm soát.

1.3. Các hệ quản trị CSDL phổ biến

Hệ quản trị	Nhà cung cấp	Ghi chú
SQL Server	Microsoft	Công cụ chính thức của khóa học (ngôn ngữ **T-SQL**, công cụ **SSMS**)
MySQL	Oracle (mã nguồn mở)	Phương án thứ hai của khóa học
PostgreSQL	Cộng đồng	Mã nguồn mở, mạnh về chuẩn SQL
Oracle Database	Oracle	Phổ biến ở hệ thống lõi của ngân hàng/chứng khoán lớn (ngôn ngữ **PL/SQL**)

Tất cả đều nói "SQL" và khoảng 80% cú pháp giống nhau. Phần khác biệt (thường là hàm và cách giới hạn số dòng) sẽ được đánh dấu **SQL Server** / **MySQL** trong bài giảng.

> **SQL Server vs MySQL:** mọi bài tập của khóa học đều phải chạy đúng trên **cả hai hệ**, nên khi học hãy ưu tiên cú pháp chung. Công cụ thống nhất ở lớp là **SQL Server + SSMS**; MySQL là phương án thay thế khi bạn không cài được SQL Server. Khi gặp cú pháp riêng, bài giảng luôn cho cả hai phiên bản.

1.4. SQL gồm những nhóm lệnh nào?

Câu lệnh SQL chia thành các nhóm theo mục đích. Khóa học đi từ "đọc" đến "ghi":

Nhóm	Viết tắt	Lệnh tiêu biểu	Học ở buổi
Truy vấn dữ liệu	DQL	`SELECT`	**1–5, 8**
Thao tác dữ liệu	DML	`INSERT`, `UPDATE`, `DELETE`	6
Định nghĩa cấu trúc	DDL	`CREATE`, `ALTER`, `DROP`	7
Điều khiển giao dịch	TCL	`COMMIT`, `ROLLBACK`	6, 10

Buổi hôm nay chỉ dùng `SELECT` – lệnh **chỉ đọc**, không làm thay đổi dữ liệu, nên bạn có thể thử thoải mái mà không sợ hỏng gì.

2. Bộ dữ liệu mẫu của khóa học

Toàn bộ khóa dùng chung một CSDL chứng khoán nhỏ tên `hnx_securities`. Trên Codelab, mỗi lần chạy bài, hệ thống nạp lại đúng bộ dữ liệu này (bản sao riêng của bạn, thoải mái `UPDATE` / `DELETE` ở các buổi sau mà không ảnh hưởng ai). Dữ liệu **không dấu** (Ha Noi, Nguyen Van An...) để tránh lỗi mã hóa giữa hai hệ. **"Hôm nay" của bộ dữ liệu cố định là 2026-09-25** – các bài chấm không dùng `GETDATE()` / `NOW()`.

> **Lưu ý:** MySQL trên Linux phân biệt chữ hoa/thường ở **tên bảng**: luôn viết đúng `Customers`, `Accounts`, `MarketPrices` ... như bên dưới.

2.1. Bức tranh tổng thể

Trước khi đi vào từng cột, hãy nhìn toàn cảnh: 9 bảng nối với nhau như thế nào. Hình dưới là **sơ đồ quan hệ (ER)** của bộ dữ liệu.

Hình minh họa: Sơ đồ ER của bộ dữ liệu mẫu (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: mỗi khung là một bảng; các đường nối là **khóa ngoại** – cột của bảng này trỏ về khóa chính của bảng kia. Đi theo đường nối từ trái sang phải là đi theo dòng nghiệp vụ: một **khách hàng** mở nhiều **tài khoản**, mỗi tài khoản đặt nhiều **lệnh**, mỗi lệnh khớp thành một hay nhiều **giao dịch (Trades)**. `Securities` là danh mục mã, được `Orders`, `Trades`, `Portfolio`, `MarketPrices` tham chiếu qua cột `symbol`.

2.2. Chi tiết từng bảng

Customers – khách hàng (40 dòng: 30 cá nhân `INDIVIDUAL`, 10 tổ chức `INSTITUTION`)

Cột	Kiểu	Ghi chú
`customer_id`	INT	**Khóa chính** (PK)
`customer_code`	NVARCHAR(10)	Mã khách hàng, duy nhất, dạng `KH100007`
`full_name`	NVARCHAR(120)	Họ tên (cá nhân) hoặc tên tổ chức
`customer_type`	NVARCHAR(12)	`INDIVIDUAL` (cá nhân) / `INSTITUTION` (tổ chức)
`birth_date`	DATE, cho phép NULL	Ngày sinh; **NULL với tổ chức**
`city`	NVARCHAR(30)	`Ha Noi` , `TP.HCM` , `Da Nang` , `Hai Phong` , `Can Tho`
`phone`	NVARCHAR(20)	Số điện thoại
`email`	NVARCHAR(100), cho phép NULL	Một số khách hàng chưa có email (NULL)
`status`	NVARCHAR(10)	`ACTIVE` / `INACTIVE` / `NEW`
`created_date`	DATE	Ngày mở hồ sơ
`referrer_id`	INT, cho phép NULL	**Khóa ngoại** (FK) tới `Customers.customer_id` – người giới thiệu

Accounts – tài khoản giao dịch (93 dòng; một khách hàng có 1–4 tài khoản)

Cột	Kiểu	Ghi chú
`account_id`	INT	**PK**
`customer_id`	INT	**FK** → `Customers.customer_id`
`account_no`	NVARCHAR(10)	Số tài khoản, duy nhất, dạng `ACC00001`
`account_type`	NVARCHAR(10)	`CASH` (tiền mặt) / `MARGIN` (ký quỹ)
`balance`	DECIMAL(18,2)	Số dư tiền (đồng)
`opened_date`	DATE	Ngày mở tài khoản
`status`	NVARCHAR(10)	`ACTIVE` / `CLOSED`

Securities – danh mục cổ phiếu (12 mã)

Cột	Kiểu	Ghi chú
`symbol`	NVARCHAR(10)	**PK** – mã chứng khoán (FPT, VCB, HPG...)
`company_name`	NVARCHAR(150)	Tên công ty
`exchange`	NVARCHAR(10)	Sàn: `HOSE` / `HNX`
`sector`	NVARCHAR(60)	Ngành
`listed_date`	DATE	Ngày niêm yết
`par_value`	INT	Mệnh giá (10.000 đ)

MarketPrices – giá thị trường theo phiên (120 dòng = 12 mã × 10 phiên, 14/09–25/09/2026)

Cột	Kiểu	Ghi chú
`symbol`, `trade_date`	NVARCHAR(10), DATE	**PK gồm hai cột**; `symbol` là FK → `Securities`
`open_price`, `high_price`, `low_price`, `close_price`	DECIMAL(12,2)	Giá mở cửa, cao nhất, thấp nhất, đóng cửa
`volume`	BIGINT	Khối lượng khớp trong phiên

Orders – lệnh đặt (120 dòng)

Cột	Kiểu	Ghi chú
`order_id`	INT	**PK**
`account_id`	INT	**FK** → `Accounts.account_id`
`symbol`	NVARCHAR(10)	**FK** → `Securities.symbol`
`side`	NVARCHAR(4)	`BUY` (mua) / `SELL` (bán)
`order_type`	NVARCHAR(2)	`LO` (lệnh giới hạn) / `MP` (lệnh thị trường)
`quantity`, `price`	INT, DECIMAL(12,2)	Khối lượng, giá đặt
`order_date`	DATE	Ngày đặt
`status`	NVARCHAR(10)	`MATCHED` (khớp hết), `PARTIAL` (khớp một phần), `PENDING` (chờ khớp), `CANCELLED` (đã hủy)

Các bảng còn lại

Bảng	Nội dung	Cột chính
Trades (86 dòng)	Các lần khớp lệnh	`trade_id` (PK), `order_id` (FK → Orders), `symbol` (FK → Securities), `quantity`, `price`, `trade_date`
Portfolio (209 dòng)	Danh mục nắm giữ	PK gồm `account_id` + `symbol` (đều là FK), `quantity` (số cổ phiếu đang giữ), `avg_cost` (giá vốn bình quân)
CustomerStats (40 dòng)	Thống kê giao dịch theo khách hàng	`customer_id` (PK, FK → Customers), `total_trade_value`, `trade_count`
AuditLog	Nhật ký đổi trạng thái khách hàng	`log_id` (PK, tự tăng), `customer_id`, `old_status`, `new_status`, `changed_at`. **Trống** ban đầu, sẽ được trigger ghi vào ở Buổi 9

2.3. Các quan hệ giữa bảng

- Một **khách hàng** có nhiều **tài khoản** (`Accounts.customer_id`).
- Một **tài khoản** có nhiều **lệnh** (`Orders.account_id`); một lệnh có thể sinh ra nhiều **lần khớp** (`Trades.order_id`).

- **Portfolio** cho biết tài khoản nào đang giữ bao nhiêu cổ phiếu mã nào.
- `Customers.referrer_id` trỏ ngược lại chính bảng `Customers` (khách hàng giới thiệu khách hàng).
- `CustomerStats` là bảng tổng hợp 1–1 với `Customers`.

> **Lưu ý:** việc ghép nhiều bảng (`JOIN`) được học ở Buổi 8. Các buổi đầu chúng ta làm việc trên **một bảng mỗi truy vấn**. Do đó câu hỏi kiểu "khách hàng Hà Nội có số dư > 50 triệu" (cần cả `Customers.city` lẫn `Accounts.balance`) tạm thời được chia thành các câu hỏi trên từng bảng; ở cuối bài có một gợi ý mở rộng.

Hãy xem thử nội dung một bảng nhỏ nhất – danh mục 12 mã chứng khoán:

```sql
SELECT * FROM Securities ORDER BY symbol;
```

```
CEO Tap doan CEO    HNX Bat dong san    2016-10-19  10000
FPT Tap doan FPT    HOSE    Cong nghe   2006-12-13  10000
HPG Tap doan Hoa Phat   HOSE    Thep    2007-11-15  10000
IDC Tong Cong ty IDICO  HNX Ha tang khu cong nghiep 2016-12-16  10000
MBS Chung khoan MB  HNX Chung khoan 2013-07-08  10000
... (12 dòng)
```

3. Công cụ và cách chạy bài trên Codelab

3.1. Công cụ làm việc

- **SQL Server Management Studio (SSMS)** – công cụ chính thức của khóa (cách cài đặt ở phần *Thực hành trên Windows* cuối buổi này). Kết nối tới server, chọn database `hnx_securities` trong hộp thả xuống ở thanh công cụ (hoặc gõ `USE hnx_securities;`), bấm **New Query**, gõ câu lệnh, bôi đen rồi nhấn **F5** (hoặc nút **Execute**). Kết quả hiện ở lưới **Results** phía dưới.
- **MySQL Workbench** – công cụ tương đương cho MySQL: mở tab SQL, nhấn **Ctrl+Enter** để chạy câu lệnh tại vị trí con trỏ.
- **DBeaver** – công cụ miễn phí kết nối được cả SQL Server, MySQL, PostgreSQL, Oracle; cũng chạy bằng **Ctrl+Enter**.

> **Mẹo:** trong lớp, mọi người dùng chung SSMS để khám phá dữ liệu (chuột phải bảng → xem 100 dòng đầu; mở nhánh **Columns** để xem kiểu dữ liệu), còn bài tập được nộp trên Codelab.

3.2. Cách chạy và nộp bài trên Codelab

1. Mở bài, đọc đề, xem bảng dữ liệu liên quan và **ví dụ mẫu** (dữ liệu gốc cùng kết quả mong đợi).
2. Chọn ngôn ngữ **SQL Server (T-SQL)** hoặc **MySQL** ở ô chọn ngôn ngữ. Bài nào cũng có lời giải cho cả hai; chọn hệ bạn đang học.
3. Viết **đúng một câu lệnh** `SELECT` trả về kết quả cần tìm (các buổi sau có bài chỉ cần `INSERT/UPDATE/...` , đề sẽ nói rõ).

4. Bấm **Nộp bài**. Codelab chạy câu lệnh của bạn trên **nhiều bộ dữ liệu khác nhau** (dữ liệu gốc và các biến thể có thêm/sửa dòng, có NULL, có giá trị đúng biên...). Kết quả được **so khớp với đáp án theo từng dòng**.

Vì so khớp theo dòng nên hãy chú ý:

- **Thứ tự cột** đúng như đề, **thứ tự dòng** đúng như yêu cầu `ORDER BY`.
- Kết quả **không in tên cột**, giá trị phân tách bằng ký tự TAB, `NULL` in là `NULL`, số tiền `DECIMAL` in đủ 2 chữ số thập phân (`150000000.00`), ngày in `yyyy-mm-dd`.

> **Cảnh báo:** đừng viết cứng kết quả theo dữ liệu gốc (ví dụ `WHERE customer_id IN (2, 4, 5)` khi đề hỏi "số dư > 100 triệu"): các bộ dữ liệu ẩn sẽ làm lời giải như vậy sai.

4. Cấu trúc cơ bản của câu lệnh SELECT

4.1. Cú pháp tối thiểu

```
SELECT cột_1, cột_2, ...
FROM tên_bảng;
```

- `SELECT` chọn **cột** nào được hiển thị (phép *chiếu*).
- `FROM` chọn **bảng** lấy dữ liệu.
- Dấu `;` kết thúc câu lệnh (T-SQL không bắt buộc nhưng nên viết; MySQL cần khi có nhiều câu lệnh).
- Từ khóa SQL không phân biệt hoa/thường; quy ước của khóa: **từ khóa VIẾT HOA**, tên bảng/cột giữ đúng như trong sơ đồ.
- **Chú thích**: `-- ghi chú một dòng` hoặc `/* ghi chú nhiều dòng */` (cả hai hệ đều hiểu). Hãy ghi chú ý định cho câu lệnh phức tạp.

Ví dụ – tên và thành phố của khách hàng:

```
SELECT customer_id, full_name, city
FROM Customers
ORDER BY customer_id;
```

```
1   Nguyen Van An   Ha Noi
2   Tran Thi Binh   Da Nang
3   Le Van Cuong    Ha Noi
4   Pham Thi Dung   Ha Noi
5   Hoang Van Em    Ha Noi
... (40 dòng)
```

Dấu `*` lấy **tất cả cột** (như ví dụ `SELECT * FROM Securities` ở mục 2) – tiện để khám phá dữ liệu, nhưng **không nên dùng trong báo cáo/ứng dụng thật**: thứ tự cột và số cột có thể đổi khi bảng thay đổi, và lấy thừa dữ liệu.

4.2. Bí danh cột và loại trùng

Bí danh cột với `AS` đổi tên cột hiển thị (rất hữu ích khi làm báo cáo):

```
SELECT full_name AS ten_khach_hang, city AS thanh_pho
FROM Customers
WHERE customer_id <= 3
ORDER BY customer_id;
```

```
Nguyen Van An   Ha Noi
Tran Thi Binh   Da Nang
Le Van Cuong    Ha Noi
```

Loại trùng với `DISTINCT`: có bao nhiêu khách hàng khác nhau sở hữu tài khoản trên 100 triệu, mỗi khách hàng chỉ kể một lần:

```
SELECT DISTINCT customer_id
FROM Accounts
WHERE balance > 100000000
ORDER BY customer_id;
```

```
2
4
5
6
7
... (27 dòng)
```

Không có `DISTINCT`, khách hàng có 3 tài khoản lớn sẽ xuất hiện 3 lần. `DISTINCT` áp dụng cho **cả bộ cột** liệt kê sau nó: `SELECT DISTINCT city, customer_type ...` loại trùng theo cặp giá trị.

> **Mẹo:** cột bí danh có khoảng trắng phải đặt trong ngoặc – `[Ten cot]` (SQL Server) hoặc `` `Ten cot` `` (MySQL). Khóa học dùng bí danh không dấu, có gạch dưới (`ten_khach_hang`) để chạy được ở cả hai hệ.

4.3. Thứ tự xử lý logic của một câu SELECT

Câu lệnh được **viết** theo thứ tự `SELECT → FROM → WHERE → ORDER BY`, nhưng hệ quản trị **hiểu và thực hiện** theo một thứ tự khác. Hình dưới là "bản đồ" thứ tự xử lý logic – bạn sẽ quay lại nó ở mọi buổi sau này.

Hình minh họa: Thứ tự xử lý của câu lệnh SELECT (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: đi từ trên xuống dưới. **`FROM`** chọn bảng nguồn trước tiên; **`WHERE`** giữ lại những dòng thỏa điều kiện; **`SELECT`** chọn cột (và đặt bí danh); **`ORDER BY`** sắp xếp kết quả cuối cùng. Các bước `GROUP BY`, `HAVING`, `TOP` / `LIMIT` trong hình sẽ được học ở Buổi 2 – hiện tại bạn chỉ cần nhớ chúng nằm ở đâu trong dòng chảy.

Hệ quả thực tế của thứ tự này:

- **Bí danh đặt ở `SELECT` không dùng được trong `WHERE`** vì `WHERE` chạy *trước* `SELECT` (lúc đó bí danh chưa tồn tại). Ngược lại `ORDER BY` chạy sau nên dùng được bí danh.
- `WHERE` loại dòng **trước** khi chọn cột, nên lọc sớm giúp hệ quản trị làm ít việc hơn.

4.4. Lọc dòng đầu tiên với WHERE

Chưa cần học hết `WHERE`, bạn đã có thể lọc theo một giá trị: thêm `WHERE cột = 'giá trị'` giữa `FROM` và `ORDER BY`. Ví dụ – chỉ lấy khách hàng **tổ chức**:

```
SELECT customer_id, customer_code, full_name, city
FROM Customers
WHERE customer_type = 'INSTITUTION'
ORDER BY customer_id;
```

```
31  KH100217    Ngan Hang TMCP Dau Tu Viet  TP.HCM
32  KH100224    Ngan Hang Nong Nghiep Phat Trien    Ha Noi
33  KH100231    Cong ty TNHH Dau tu Minh Phat   Ha Noi
34  KH100238    Quy Dau tu Bao Viet TP.HCM
35  KH100245    Cong ty CP Chung khoan An Binh  Ha Noi
... (10 dòng)
```

Chuỗi đặt trong **nháy đơn**, so sánh chuỗi mặc định **không phân biệt hoa/thường** trên cả hai hệ. Mục 5 sẽ đi sâu vào `WHERE`.

Thực hành: vừa học xong `SELECT`, `FROM`, bí danh và một điều kiện `WHERE` đơn giản, hãy làm **bài 1.1** để lấy danh sách toàn bộ khách hàng rồi lọc riêng nhóm khách hàng tổ chức.

- **Bài 1.1** — Danh sách toàn bộ khách hàng *(độ khó: dễ)*
- **Bài 1.1** — Khách hàng tổ chức *(độ khó: dễ)*

5. Mệnh đề FROM và WHERE

5.1. Chọn bảng dữ liệu

Việc đầu tiên khi gặp một câu hỏi nghiệp vụ là **chọn đúng bảng**:

Câu hỏi	Bảng
Ai là khách hàng, ở đâu, loại gì?	`Customers`
Tài khoản nào, số dư bao nhiêu?	`Accounts`
Mã nào niêm yết ở sàn nào?	`Securities`
Giá đóng cửa từng phiên?	`MarketPrices`
Lệnh mua/bán nào đã đặt, đã khớp?	`Orders`, `Trades`

5.2. Lọc dòng với WHERE và các toán tử so sánh

`WHERE` chỉ giữ lại những dòng thỏa điều kiện.

Toán tử	Ý nghĩa	Ví dụ
`=`	bằng	`city = 'Ha Noi'`
`<>` hoặc `!=`	khác	`status <> 'ACTIVE'`
`<`, `>`	nhỏ hơn, lớn hơn	`balance > 100000000`
`<=`, `>=`	nhỏ hơn/bằng, lớn hơn/bằng	`quantity >= 1000`

> **Lưu ý:** trong chuẩn SQL, "khác" viết là `<>`; `!=` cũng chạy được trên cả SQL Server và MySQL. Khóa học dùng `<>` cho gọn nhất quán.

Quy tắc viết giá trị:

- **Chuỗi** và **ngày** đặt trong dấu nháy đơn: `'Ha Noi'`, `'2026-09-25'` (ngày dạng `yyyy-mm-dd` chạy đúng trên cả hai hệ).
- **Số** không có nháy, **không có dấu phân cách hàng nghìn**: `100000000`, không phải `100.000.000` hay `100,000,000`. Để dễ đọc, nhớ 100 triệu = 1 và 8 số 0.

Ví dụ – tài khoản có số dư lớn hơn 100 triệu:

```sql
SELECT account_id, customer_id, balance
FROM Accounts
WHERE balance > 100000000
ORDER BY account_id;
```

```
4   2   8600000000.00
8   4   230000000.00
9   4   1250000000.00
11  5   3400000000.00
12  6   780000000.00
... (46 dòng)
```

Ví dụ – khách hàng ở Hà Nội (dữ liệu không dấu nên viết `'Ha Noi'`), và khách hàng mở hồ sơ từ đầu tháng 9/2026:

```sql
SELECT customer_id, full_name, created_date
FROM Customers
WHERE created_date >= '2026-09-01'
ORDER BY created_date, customer_id;
```

```
19  Cao Van Yen 2026-09-14
12  Ly Thi Quynh    2026-09-19
5   Hoang Van Em    2026-09-23
26  Hoang Thi Huong 2026-09-23
```

> **SQL Server vs MySQL:** so sánh chuỗi mặc định **không phân biệt hoa/thường** trên cả hai hệ (collation `CI`), nên `'ha noi'` cũng khớp `'Ha Noi'`. Tuy vậy hãy viết đúng hoa/thường như dữ liệu. Chuỗi Unicode có dấu trong SQL Server nên viết `N'Hà Nội'` (tiền tố `N`); dữ liệu khóa học không dấu nên không cần.

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
`city = Ha Noi` (quên nháy)	Lỗi cú pháp	`city = 'Ha Noi'`
`city = "Ha Noi"` (nháy kép)	MySQL chấp nhận, SQL Server hiểu là tên cột → lỗi	Luôn dùng nháy đơn
`city = 'Hà Nội'` (có dấu)	Không ra dòng nào vì dữ liệu không dấu	`city = 'Ha Noi'`
`balance > 100.000.000`	Lỗi cú pháp/sai giá trị	`balance > 100000000`

5.3. Kết hợp nhiều điều kiện: AND, OR, NOT

Toán tử	Dòng được giữ khi...
`A AND B`	**cả hai** điều kiện đúng
`A OR B`	**ít nhất một** điều kiện đúng
`NOT A`	điều kiện `A` **sai**

Ví dụ – khách hàng cá nhân, đang hoạt động, ở Hà Nội:

```sql
SELECT customer_id, full_name, city
FROM Customers
WHERE customer_type = 'INDIVIDUAL' AND status = 'ACTIVE' AND city = 'Ha Noi'
ORDER BY customer_id;
```

```
1   Nguyen Van An   Ha Noi
3   Le Van Cuong    Ha Noi
7   Dang Van Khoa   Ha Noi
8   Bui Thi Lan Ha Noi
10  Ngo Thi Nga Ha Noi
... (12 dòng)
```

Độ ưu tiên: `NOT` → `AND` → `OR`. `AND` "dính" chặt hơn `OR`, giống phép nhân so với phép cộng. Khi trộn hai loại, **luôn dùng ngoặc** để thể hiện ý định. So sánh hai câu sau, cùng nhằm tìm tài khoản *đang ACTIVE* thuộc loại `MARGIN` hoặc có số dư trên 1 tỷ:

```sql
-- SAI (thiếu ngoặc): hiểu là (ACTIVE và MARGIN) HOẶC (số dư > 1 tỷ)
SELECT COUNT(*) AS so_tai_khoan FROM Accounts
WHERE status = 'ACTIVE' AND account_type = 'MARGIN' OR balance > 1000000000;
```

```
45
```

```sql
-- ĐÚNG: ACTIVE và (MARGIN hoặc số dư > 1 tỷ)
SELECT COUNT(*) AS so_tai_khoan FROM Accounts
WHERE status = 'ACTIVE' AND (account_type = 'MARGIN' OR balance > 1000000000);
```

```
44
```

Chênh lệch 1 dòng chính là tài khoản `ACC00009` – loại `MARGIN`, số dư 1,25 tỷ nhưng đã `CLOSED` – lọt vào kết quả của câu sai.

> **Cảnh báo:** trong báo cáo rủi ro, một tài khoản đã đóng vẫn lọt vào danh sách theo dõi là loại sai sót khó phát hiện vì câu lệnh vẫn chạy "bình thường". Thiếu ngoặc không báo lỗi – nó chỉ trả sai kết quả.

`NOT` đảo điều kiện: `WHERE NOT city = 'Ha Noi'` tương đương `WHERE city <> 'Ha Noi'`. `NOT` hay dùng nhất ở dạng `NOT IN`, `NOT LIKE`, `NOT BETWEEN`, `IS NOT NULL`.

Thực hành: vừa học xong `WHERE`, các toán tử so sánh và `AND` / `OR` / `NOT`, hãy làm **bài 1.2** để tìm tài khoản số dư lớn, khách hàng theo thành phố và lập danh sách tài khoản cần theo dõi (chú ý ngoặc!).

- **Bài 1.2** — Tài khoản có số dư trên 100 triệu *(độ khó: dễ)*
- **Bài 1.2** — Khách hàng ở Hà Nội *(độ khó: dễ)*
- **Bài 1.2** — Tài khoản cần theo dõi (AND/OR) *(độ khó: trung bình)*

5.4. LIKE, IN, BETWEEN

LIKE – so khớp mẫu chuỗi

Ký tự đại diện	Ý nghĩa
`%`	chuỗi bất kỳ (kể cả rỗng)
`_`	đúng một ký tự

Mẫu	Khớp với	Không khớp
`'Nguyen%'`	`Nguyen Van An`, `Nguyen Thi Chi`	`Tran Thi Binh`
`'%Ngan Hang%'`	`Ngan Hang TMCP Dau Tu Viet`	`Quy Dau tu Bao Viet`
`'KH10002_'`	`KH100021`, `KH100028`	`KH100014`, `KH100035`

```sql
-- Họ Nguyen
SELECT customer_id, full_name FROM Customers WHERE full_name LIKE 'Nguyen%' ORDER BY
        customer_id;
```

```
1   Nguyen Van An
22  Nguyen Thi Chi
```

```sql
-- Tên khách hàng có chứa "Ngan Hang"
SELECT customer_id, full_name
FROM Customers
WHERE full_name LIKE '%Ngan Hang%'
ORDER BY customer_id;
```

```
31  Ngan Hang TMCP Dau Tu Viet
32  Ngan Hang Nong Nghiep Phat Trien
37  Ngan Hang Quan Doi Mien Bac
40  Ngan Hang Ban Le Sai Gon
```

> **Lưu ý:** mẫu bắt đầu bằng `%` (`'%Ngan Hang%'`) buộc hệ quản trị đọc toàn bảng, không tận dụng được chỉ mục (Buổi 10). Với bảng vài triệu dòng hãy cân nhắc. Muốn tìm đúng ký tự `%` hay `_` thì phải "thoát" (escape) – tình huống hiếm, sẽ không gặp trong khóa học.

> **SQL Server vs MySQL: SQL Server** còn hỗ trợ lớp ký tự `[...]` trong `LIKE` (`LIKE '[NT]%'`) mà MySQL không có – vì vậy bài của khóa dùng `OR` thay vì lớp ký tự.

IN – thuộc một danh sách

```sql
SELECT customer_id, customer_code, full_name
FROM Customers
WHERE customer_code IN ('KH100007', 'KH100056', 'KH100224')
ORDER BY customer_code;
```

```
1   KH100007    Nguyen Van An
8   KH100056    Bui Thi Lan
32  KH100224    Ngan Hang Nong Nghiep Phat Trien
```

`IN (a, b, c)` tương đương `= a OR = b OR = c` nhưng gọn hơn nhiều. Dạng phủ định `NOT IN` dùng để loại trừ – ví dụ đếm khách hàng ở tỉnh khác Hà Nội và TP.HCM:

```sql
SELECT COUNT(*) AS so_khach_hang_tinh_khac
FROM Customers
WHERE city NOT IN ('Ha Noi', 'TP.HCM');
```

```
11
```

BETWEEN – trong một khoảng, bao gồm cả hai đầu mút

```sql
SELECT customer_id, full_name, created_date
FROM Customers
WHERE created_date BETWEEN '2021-01-01' AND '2021-12-31'
ORDER BY created_date;
```

```
14  Phan Thi Thao   2021-02-21
32  Ngan Hang Nong Nghiep Phat Trien    2021-04-17
22  Nguyen Thi Chi  2021-05-30
1   Nguyen Van An   2021-06-29
... (9 dòng)
```

`x BETWEEN a AND b` = `x >= a AND x <= b`. Với ngày, `a` phải nhỏ hơn hoặc bằng `b` (đảo ngược sẽ không cho dòng nào). Với `NOT BETWEEN` ta lấy phần ngoài khoảng.

> **Cảnh báo:** `BETWEEN` gồm cả hai đầu – số dư đúng bằng 100 triệu **có** thuộc `BETWEEN 100000000 AND 500000000`, còn `> 100000000` thì không. Khi đề nói "từ … đến …" (bao gồm) dùng `BETWEEN`; khi đề nói "lớn hơn" (hẳn) dùng `>`.

5.5. NULL – giá trị "không biết"

`NULL` không phải số 0 hay chuỗi rỗng; nó nghĩa là "chưa có / không xác định". Ở bộ dữ liệu: `email` của 5 khách hàng là `NULL`, `birth_date` của tổ chức là `NULL`, `referrer_id` của phần lớn khách hàng là `NULL`.

Mọi so sánh với `NULL` bằng `=`, `<>`, `>`... đều **không đúng cũng không sai** (kết quả "không biết"), và dòng bị loại. Cần dùng `IS NULL` / `IS NOT NULL`:

```
SELECT customer_id, full_name, email
FROM Customers
WHERE email IS NULL
ORDER BY customer_id;
```

```
6   Vu Thi Hoa  NULL
12  Ly Thi Quynh    NULL
18  Ha Thi Xuan NULL
24  Le Thi Giang    NULL
30  Do Thi Nhung    NULL
```

Bẫy kinh điển: so sánh `= NULL` **không bao giờ** cho dòng nào, và `<>` cũng loại luôn các dòng NULL:

```
SELECT COUNT(*) AS so_dong FROM Customers WHERE email = NULL;
```

```
0
```

```
-- referrer_id <> 1: KHÔNG bao gồm khách chưa có người giới thiệu (NULL)
SELECT COUNT(*) AS so_dong FROM Customers WHERE referrer_id <> 1;
```

```
6
```

```
-- Muốn "khác 1 hoặc chưa có người giới thiệu" phải nói rõ
SELECT COUNT(*) AS so_dong FROM Customers WHERE referrer_id <> 1 OR referrer_id IS NULL;
```

```
38
```

> **Ví dụ nghiệp vụ:** phòng Vận hành chuẩn bị gửi thông báo qua email cho toàn bộ khách hàng. Viết `WHERE email <> ''` sẽ **bỏ sót** nhầm khách chưa có email hoặc cho kết quả khó đoán; cách đúng là `WHERE email IS NOT NULL` (và cân nhắc thêm chuỗi rỗng khi dữ liệu nhập tay).

Thực hành: vừa học xong `LIKE`, `IN`, `BETWEEN` và `NULL`, hãy làm **bài 1.3** – tìm tên chứa "Ngan Hang", tra cứu theo danh sách mã khách hàng, tài khoản trong khoảng số dư và khách hàng chưa có email.

- **Bài 1.3** — Khách hàng có tên chứa "Ngan Hang" *(độ khó: dễ)*
- **Bài 1.3** — Tra cứu theo danh sách mã khách hàng *(độ khó: dễ)*
- **Bài 1.3** — Tài khoản số dư từ 100 đến 500 triệu *(độ khó: trung bình)*
- **Bài 1.3** — Khách hàng chưa có email *(độ khó: trung bình)*

6. Mệnh đề ORDER BY

Nếu không có `ORDER BY`, thứ tự dòng trả về **không được đảm bảo** (có thể đúng theo thứ tự nhập nhưng không có gì cam kết). Muốn kết quả ổn định – đặc biệt khi chấm bài tự động – phải có `ORDER BY`, và nó luôn nằm **cuối câu lệnh**.

6.1. ASC và DESC

- `ASC` (mặc định): tăng dần – số nhỏ → lớn, chữ A → Z, ngày cũ → mới.
- `DESC`: giảm dần – số lớn → nhỏ, ngày mới → cũ.

```
SELECT account_id, balance
FROM Accounts
WHERE account_id IN (1, 2, 3, 4)
ORDER BY balance DESC;
```

```
4   8600000000.00
1   67000000.00
3   32000000.00
2   8000000.00
```

NULL khi sắp xếp: cả SQL Server và MySQL coi `NULL` là **nhỏ nhất** – nằm đầu khi `ASC`, cuối khi `DESC`. Ví dụ `ORDER BY birth_date` sẽ đặt các tổ chức (ngày sinh NULL) lên trước:

```
SELECT customer_id, full_name, birth_date
FROM Customers
ORDER BY birth_date, customer_id;
```

```
31  Ngan Hang TMCP Dau Tu Viet  NULL
32  Ngan Hang Nong Nghiep Phat Trien    NULL
33  Cong ty TNHH Dau tu Minh Phat   NULL
34  Quy Dau tu Bao Viet NULL
... (40 dòng)
```

6.2. Sắp xếp theo nhiều cột và thứ tự ưu tiên

```
ORDER BY cột_1 [ASC|DESC], cột_2 [ASC|DESC], ...
```

Cột đứng trước có ưu tiên cao hơn. Hệ quản trị sắp theo `cột_1`; **chỉ khi hai dòng bằng nhau ở `cột_1`** mới xét tới `cột_2`, và cứ thế. Mỗi cột có chiều `ASC` / `DESC` riêng.

```sql
SELECT customer_id, full_name, city, created_date
FROM Customers
WHERE city IN ('Ha Noi', 'TP.HCM') AND customer_type = 'INDIVIDUAL'
ORDER BY city, created_date DESC, customer_id;
```

```
5   Hoang Van Em    Ha Noi  2026-09-23
26  Hoang Thi Huong Ha Noi  2026-09-23
19  Cao Van Yen Ha Noi  2026-09-14
25  Pham Van Hai    Ha Noi  2025-07-05
28  Dang Thi Linh   Ha Noi  2025-01-02
... (19 dòng)
```

Đọc kết quả: các dòng gom theo thành phố (A → Z); trong cùng thành phố, khách mở hồ sơ **mới nhất** lên đầu; hai khách cùng ngày (5 và 26 cùng 2026-09-23) thì xếp theo `customer_id`.

> **Mẹo:** luôn thêm khóa duy nhất (như `customer_id`) làm cột sắp xếp cuối cùng để kết quả *tất định* – hai lần chạy cho cùng một thứ tự – điều bắt buộc khi kết quả được so khớp tự động.

Đổi thứ tự cột trong `ORDER BY` là đổi ưu tiên: `ORDER BY created_date DESC, city` sẽ cho một danh sách hoàn toàn khác.

> **SQL Server vs MySQL:** cú pháp `ORDER BY` giống nhau. Cả hai cho phép `ORDER BY 2 DESC` (theo số thứ tự cột trong `SELECT`), nhưng cách này dễ sai khi sửa danh sách cột; ưu tiên dùng tên cột. Riêng việc **giới hạn số dòng** (lấy 10 dòng đầu) khác nhau giữa hai hệ – sẽ học ở Buổi 2.

Thực hành: vừa học xong `ORDER BY` với nhiều cột và thứ tự ưu tiên, hãy làm **bài 1.4** – từ sắp xếp lệnh bán, xếp hạng tài khoản đến bài tổng hợp lập danh sách khách hàng chăm sóc.

- **Bài 1.4** — Lệnh bán đã khớp gần đây *(độ khó: trung bình)*
- **Bài 1.4** — Xếp hạng tài khoản theo loại và số dư *(độ khó: trung bình)*
- **Bài 1.4** — Danh sách khách hàng chăm sóc *(độ khó: khó)*

7. Áp dụng: từ yêu cầu nghiệp vụ đến câu lệnh

Yêu cầu: *"Lấy các lệnh **mua** đã **khớp hết**, khối lượng **từ 1.000** trở lên, đặt trong tháng 9/2026; mới nhất lên trước; cùng ngày thì khối lượng lớn xếp trước, cùng khối lượng thì theo mã lệnh."*

Bóc tách: bảng `Orders`; điều kiện `side = 'BUY'` AND `status = 'MATCHED'` AND `quantity >= 1000` AND ngày trong khoảng; sắp `order_date DESC, quantity DESC, order_id`.

```sql
SELECT order_id, account_id, symbol, quantity, price, order_date
FROM Orders
WHERE side = 'BUY' AND status = 'MATCHED' AND quantity >= 1000
  AND order_date BETWEEN '2026-09-01' AND '2026-09-30'
ORDER BY order_date DESC, quantity DESC, order_id;
```

```
72  53  NVB 1000    9000.00 2026-09-25
2   80  MBS 2000    25000.00    2026-09-24
38  30  VCS 1000    61000.00    2026-09-24
53  2   SHS 1000    15500.00    2026-09-22
39  53  VCB 1000    92500.00    2026-09-21
... (10 dòng)
```

Quy trình 4 bước bạn nên áp dụng cho mọi bài: **(1)** xác định bảng → **(2)** liệt kê từng điều kiện, nối bằng `AND` / `OR`, thêm ngoặc → **(3)** chọn cột theo đúng thứ tự đề yêu cầu → **(4)** viết `ORDER BY` có khóa phụ. Sau cùng hãy tự hỏi: "nếu có dòng NULL, dòng đúng biên, dòng đồng hạng thì sao?"

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Quên nháy đơn quanh chuỗi/ngày, hoặc dùng nháy kép	Lỗi cú pháp hoặc SQL Server hiểu là tên cột	Luôn dùng nháy đơn: `'Ha Noi'`, `'2026-09-25'`
Trộn `AND` / `OR` không có ngoặc	Không báo lỗi nhưng sai kết quả (lọt dòng không mong muốn)	Luôn thêm ngoặc: `A AND (B OR C)`
So sánh với NULL bằng `=` / `<>`	Kết quả rỗng hoặc thiếu dòng	Dùng `IS NULL` / `IS NOT NULL`
Quên `BETWEEN` gồm cả hai đầu mút	Lệch một số dòng đúng biên	Kiểm tra lại đề "từ … đến …" hay "lớn hơn"
Dùng bí danh ở `SELECT` trong `WHERE`	Lỗi `Invalid column name`	Lặp lại biểu thức gốc, hoặc dùng bí danh ở `ORDER BY`
Không có `ORDER BY` (hoặc thiếu khóa phụ)	Thứ tự dòng không đảm bảo, chấm sai	`ORDER BY` luôn có khóa duy nhất ở cuối
Dùng `SELECT *` trong báo cáo	Thừa cột, vỡ khi bảng đổi cấu trúc	Liệt kê đúng các cột cần

Tóm tắt

Nội dung	Cú pháp / ghi nhớ
Lấy cột từ bảng	`SELECT cột FROM bảng` (tránh `*` trong báo cáo)
Bí danh, loại trùng	`cột AS tên_mới` , `SELECT DISTINCT ...`
Thứ tự xử lý	`FROM` → `WHERE` → `SELECT` → `ORDER BY`
Lọc dòng	`WHERE điều_kiện` (chuỗi/ngày trong nháy đơn, số không có dấu phân cách)
So sánh	`=` , `<>` / `!=` , `<` , `>` , `<=` , `>=`
Logic	`NOT` > `AND` > `OR` ; luôn dùng ngoặc khi trộn
Mẫu chuỗi	`LIKE '%…%'` , `%` nhiều ký tự, `_` một ký tự
Tập giá trị / khoảng	`IN (…)` , `BETWEEN a AND b` (gồm hai đầu)
NULL	`IS NULL` , `IS NOT NULL` ; `= NULL` luôn sai
Sắp xếp	`ORDER BY c1 ASC/DESC, c2 …` – cột trước ưu tiên hơn; thêm khóa phụ
Khác biệt hai hệ ở buổi này	Gần như không có; lưu ý tên bảng phân biệt hoa/thường trên MySQL, `N'...'` cho chuỗi Unicode trong SQL Server

Gợi ý mở rộng

- Trong SSMS, mở **Object Explorer** → **Databases** → **hnx_securities** → **Tables** → **Columns** để đối chiếu kiểu dữ liệu với sơ đồ ở mục 2; thử `SELECT * FROM` từng bảng để quen dữ liệu.
- Thử "khách hàng Hà Nội có tài khoản số dư > 50 triệu" bằng một cách nhìn trước Buổi 5 (truy vấn con):

  ```sql
  SELECT customer_id, full_name
  FROM Customers
  WHERE city = 'Ha Noi'
    AND customer_id IN (SELECT customer_id FROM Accounts WHERE balance > 50000000)
  ORDER BY customer_id;
  ```

  Ở Buổi 8, cùng yêu cầu này sẽ được giải bằng `JOIN` .
- Tự đặt câu hỏi nghiệp vụ trên bảng `Securities` (mã nào thuộc sàn HNX, ngành ngân hàng?) và `MarketPrices` (những phiên nào giá đóng cửa thấp hơn giá mở cửa?) rồi viết `SELECT` cho từng câu.
- Đọc thêm: cách SQL Server và MySQL xử lý `NULL` khi sắp xếp, và collation quyết định việc phân biệt hoa/thường như thế nào.

Bài tập thực hành của buổi

Danh sách đầy đủ các bài của buổi (bạn đã gặp từng nhóm ngay trong bài giảng), nộp trên Codelab bằng SQL Server hoặc MySQL:

- **Bài 1.1** — Danh sách toàn bộ khách hàng *(độ khó: dễ)*
- **Bài 1.1** — Khách hàng tổ chức *(độ khó: dễ)*
- **Bài 1.2** — Tài khoản có số dư trên 100 triệu *(độ khó: dễ)*
- **Bài 1.2** — Khách hàng ở Hà Nội *(độ khó: dễ)*
- **Bài 1.2** — Tài khoản cần theo dõi (AND/OR) *(độ khó: trung bình)*
- **Bài 1.3** — Khách hàng có tên chứa "Ngan Hang" *(độ khó: dễ)*
- **Bài 1.3** — Tra cứu theo danh sách mã khách hàng *(độ khó: dễ)*
- **Bài 1.3** — Tài khoản số dư từ 100 đến 500 triệu *(độ khó: trung bình)*
- **Bài 1.3** — Khách hàng chưa có email *(độ khó: trung bình)*
- **Bài 1.4** — Lệnh bán đã khớp gần đây *(độ khó: trung bình)*
- **Bài 1.4** — Xếp hạng tài khoản theo loại và số dư *(độ khó: trung bình)*
- **Bài 1.4** — Danh sách khách hàng chăm sóc *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: cài đặt và làm quen công cụ

Phần này dành cho bạn **tự làm trên máy Windows cá nhân** (ngoài 180 phút trên lớp, không chấm tự động). Mục tiêu: có sẵn một "phòng thực hành" **SQL Server + SSMS (công cụ chính thức của khóa, dùng thống nhất ở lớp)** để từ Buổi 2 trở đi bạn thử mọi ví dụ trên máy mình. MySQL + Workbench chỉ là phương án thay thế khi máy không cài được SQL Server. Nếu chưa cài được, có thể dùng nút **Chạy thử** và công cụ Chạy SQL trên Codelab để làm các ví dụ.

> **Cấu hình khuyến nghị:** Windows 10/11 64-bit, RAM từ 8 GB, trống khoảng 10 GB ổ đĩa, tài khoản có quyền quản trị (Administrator) khi cài. Hãy cài **SQL Server + SSMS** trước; phần MySQL (mục C) chỉ làm khi cần.

A. SQL Server Express + SSMS

Bước 1. Cài SQL Server Express (bộ máy cơ sở dữ liệu).

1. Tải **SQL Server Express** (miễn phí) từ trang tải SQL Server của Microsoft; chọn bản Express của phiên bản mới nhất (2022 trở lên).
2. Chạy file vừa tải, chọn kiểu cài **Basic**, đồng ý điều khoản, giữ thư mục mặc định rồi bấm **Install**.
3. Khi cài xong, hộp thoại hiển thị **tên instance** và **chuỗi kết nối**. Hãy ghi lại tên instance (mặc định là `localhost\SQLEXPRESS`).

> **Lưu ý:** kiểu Basic tự thêm tài khoản Windows đang cài vào nhóm quản trị (sysadmin) của SQL Server, nên bạn đăng nhập bằng **Windows Authentication**, không cần mật khẩu `sa`.

Bước 2. Cài SQL Server Management Studio (SSMS).

1. Cũng ở hộp thoại cuối của bước 1 có nút **Install SSMS**; hoặc tìm "SSMS download" trên trang Microsoft Learn và tải bản mới nhất.
2. Chạy trình cài đặt, chấp nhận mặc định, chờ hoàn tất. Nếu được yêu cầu khởi động lại Windows, hãy khởi động lại.
3. Mở **SQL Server Management Studio** từ menu Start.

Bước 3. Kết nối tới server.

Hộp thoại **Connect to Server** hiện ra khi mở SSMS:

Ô	Giá trị
Server type	Database Engine
Server name	`localhost\SQLEXPRESS` (đúng tên instance đã ghi ở bước 1)
Authentication	Windows Authentication
Trust server certificate	**Tick** chọn (máy cá nhân dùng chứng chỉ tự ký, nếu không tick sẽ báo lỗi chuỗi chứng chỉ)

Bấm **Connect**. Bên trái xuất hiện cây **Object Explorer** với tên server; mở nhánh **Databases** sẽ thấy các CSDL hệ thống.

Bước 4. Nạp bộ dữ liệu `hnx_securities`.

1. Tải file `hnx_securities_sqlserver.sql` ở trang **Tài liệu & môi trường thực hành** của khóa (đường dẫn "Tài liệu & hướng dẫn cài đặt môi trường" ở đầu trang môn SQL), lưu vào thư mục dễ nhớ, ví dụ `D:\SQL\`.
2. Trong SSMS: **File** → **Open** → **File...** chọn file vừa tải.
3. Bấm **Execute** (hoặc **F5**). Script tự xóa bản cũ (nếu có), tạo database `hnx_securities` và nạp dữ liệu; chờ vài giây đến khi thấy dòng *Completion time* ở tab **Messages**.
4. Trong **Object Explorer**, bấm chuột phải vào **Databases** → **Refresh**, sẽ thấy `hnx_securities`.

Bước 5. Chạy truy vấn đầu tiên và tự kiểm tra.

1. Bấm chuột phải vào `hnx_securities` → **New Query** (hoặc nhấn **Ctrl+N**). Cửa sổ truy vấn mới tự gắn với database này; xem tên database ở hộp thả xuống trên thanh công cụ.
2. Gõ đoạn sau và nhấn **F5**:

```
USE hnx_securities;
SELECT COUNT(*) AS so_khach_hang FROM Customers;   -- phải ra 40
SELECT COUNT(*) AS so_tai_khoan   FROM Accounts;    -- phải ra 93
SELECT COUNT(*) AS so_lenh        FROM Orders;      -- phải ra 120
```

3. Mỗi câu `SELECT` cho một lưới kết quả riêng ở tab **Results**. Nếu ba số khớp 40, 93, 120 là môi trường đã đúng.

B. Các thao tác cơ bản trong SSMS cần thuộc

Việc cần làm	Cách thực hiện
Mở cửa sổ truy vấn mới	**Ctrl+N** hoặc nút **New Query**
Chạy	**F5** / **Ctrl+E**. Nếu có **bôi đen** thì chỉ chạy phần bôi đen, không có thì chạy cả cửa sổ
Đổi database đang dùng	Hộp thả xuống trên thanh công cụ hoặc gõ `USE hnx_securities;`
Xem cấu trúc bảng	Object Explorer → mở rộng **Tables** → bảng → **Columns**
Xem nhanh dữ liệu bảng	Chuột phải bảng → **Select Top 1000 Rows**
Chú thích / bỏ chú thích	Bôi đen rồi **Ctrl+K, Ctrl+C** / **Ctrl+K, Ctrl+U**
Lưu câu lệnh	**Ctrl+S**, lưu file `.sql` (nên lưu UTF-8 để giữ tiếng Việt có dấu)
Chuyển giữa lưới / văn bản / file	**Ctrl+D** (lưới), **Ctrl+T** (văn bản), **Ctrl+Shift+F** (ra file)

> **Cảnh báo:** nếu bạn **bôi đen một phần** câu lệnh rồi bấm F5, SSMS chỉ chạy phần bôi đen. Quên bôi cả mệnh đề `WHERE` là nguyên nhân kinh điển gây `UPDATE` / `DELETE` nhầm toàn bảng (xem Buổi 6). Thói quen tốt: nhấn **Ctrl+A** hoặc bỏ bôi đen trước khi chạy một khối nhiều dòng.

C. Phương án phụ: MySQL Server + Workbench

1. Tải **MySQL Installer for Windows** (gói `mysql-installer-community`) từ `dev.mysql.com/downloads/installer`.
2. Chạy, chọn **Custom** và tick **MySQL Server 8.x** cùng **MySQL Workbench** (hoặc chọn **Developer Default** nếu máy rộng chỗ). Bấm **Next** → **Execute** để cài.
3. Ở bước cấu hình giữ nguyên: *Development Computer*, cổng **3306**, *Use Strong Password Encryption*. Ở mục **Accounts and Roles** đặt mật khẩu cho tài khoản `root` (ghi nhớ nó) và để dịch vụ chạy cùng Windows (*Start the MySQL Server at System Startup*).
4. Mở **MySQL Workbench**, bấm vào kết nối **Local instance MySQL...** ở màn hình chính, nhập mật khẩu `root`.
5. **File** → **Open SQL Script...** chọn `hnx_securities_mysql.sql`, bấm biểu tượng **tia chớp** (không bôi đen = chạy cả script).
6. Mở tab truy vấn mới (**Ctrl+T**), gõ và chạy bằng **Ctrl+Enter** (chạy câu tại con trỏ) hoặc **Ctrl+Shift+Enter** (chạy toàn bộ / phần bôi đen):

```sql
USE hnx_securities;
SELECT COUNT(*) FROM Customers;   -- 40
```

> **Lưu ý riêng của Windows:** MySQL cài trên Windows mặc định **không phân biệt hoa/thường tên bảng** (`customers` và `Customers` là một), trong khi MySQL trên Linux của Codelab thì có. Khi làm bài nộp lên Codelab vẫn phải viết đúng `Customers`, `MarketPrices` ...

D. Xử lý sự cố thường gặp

Hiện tượng	Nguyên nhân / Cách xử lý
SSMS báo *A connection was successfully established... certificate chain was issued by an authority that is not trusted*	Tick **Trust server certificate** trong hộp thoại Connect
Không kết nối được `localhost\SQLEXPRESS`	Nhấn **Win+R**, gõ `services.msc`, tìm **SQL Server (SQLEXPRESS)**, bảo đảm trạng thái *Running*; nếu không thì chuột phải → **Start**
Chữ tiếng Việt trong file `.sql` bị lỗi font	Mở file bằng SSMS (không dùng Notepad cũ) và lưu lại với bảng mã **UTF-8**
Workbench báo *Can't connect to MySQL server on 'localhost'*	Kiểm tra dịch vụ **MySQL80** (hoặc **MySQL84**) trong `services.msc` đã *Running* chưa
Chạy lại script nạp dữ liệu có bị trùng không?	Không: script xóa database `hnx_securities` cũ rồi tạo mới, nên chạy lại là cách **đưa dữ liệu về trạng thái ban đầu**

E. Bài tự làm trên máy

1. Cài xong và đạt ba số 40, 93, 120 ở bước 5.
2. Mở lại **bài 1.1** (hoặc bất kỳ bài nào của Buổi 1), tự viết lời giải trong SSMS/Workbench, so với kết quả mong đợi trong đề rồi mới nộp trên Codelab.
3. Chụp màn hình lưới kết quả của một truy vấn `SELECT ... WHERE ... ORDER BY ...` do bạn tự đặt, để dùng làm báo cáo cuối khóa (không bắt buộc).

Buổi 2 — Sắp xếp dữ liệu và các hàm thống kê cơ bản

Về mục lục

Buổi 1 giúp chúng ta *lấy ra từng dòng* thỏa điều kiện. Buổi này chuyển sang câu hỏi mà lãnh đạo và bộ phận phân tích quan tâm hơn: **"tổng cộng bao nhiêu?", "trung bình bao nhiêu?", "top 10 là ai?", "mỗi thành phố/mỗi mã có bao nhiêu?"**. Đó là lúc chúng ta dùng các **hàm thống kê (aggregate functions)**, `GROUP BY`, `HAVING` và cách giới hạn/phân trang kết quả – nền tảng của mọi báo cáo số liệu trong công ty chứng khoán.

> **Lưu ý:** cơ sở dữ liệu mẫu (9 bảng `Customers`, `Accounts`, `Securities`, `MarketPrices`, `Orders`, `Trades`, `Portfolio`, `CustomerStats`, `AuditLog`) đã được mô tả đầy đủ, kèm sơ đồ quan hệ, ở mục **"Bộ dữ liệu mẫu của khóa học" trong bài giảng Buổi 1**. Hôm nay chủ yếu dùng `Customers`, `Accounts`, `Orders`, `Securities`, `MarketPrices`. "Hôm nay" của bộ dữ liệu là 2026-09-25. Vì **chưa học `JOIN`** (Buổi 8), các bài "tài sản khách hàng" được tính từ bảng `Accounts` bằng `GROUP BY customer_id`.

Mục tiêu buổi học

Sau buổi này bạn có thể:

- Sắp xếp kết quả tất định với `ORDER BY` nhiều cột, khóa phụ và biểu thức.
- Dùng `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`; phân biệt `COUNT(*)` và `COUNT(cột)`; hiểu cách xử lý NULL và chia số nguyên ở hai hệ.
- Giới hạn và phân trang kết quả: `TOP`, `OFFSET ... FETCH` (SQL Server) và `LIMIT ... OFFSET` (MySQL); làm bài "Top 10 khách hàng giàu nhất".
- Thống kê theo nhóm với `GROUP BY` (một hoặc nhiều cột) và lọc nhóm bằng `HAVING`; phân biệt `WHERE` với `HAVING`.
- Dùng `COUNT(DISTINCT)`, `STRING_AGG` / `GROUP_CONCAT`, `ROUND`, `CAST` để làm báo cáo thống kê chính xác và giống nhau trên hai hệ quản trị.

Bản đồ buổi học

Phần	Nội dung	Hình thức	Thời lượng
1	Ôn tập và mở rộng `ORDER BY` + **bài 2.1**	5 phút lý thuyết + 18 phút thực hành	23 phút
2	Hàm thống kê `COUNT` / `SUM` / `AVG` / `MIN` / `MAX` + **bài 2.2**	11 phút lý thuyết + 24 phút thực hành	35 phút
3–4	`TOP` / `LIMIT` / `OFFSET`, `GROUP BY`, ứng dụng Top 10 + **bài 2.3**	18 phút lý thuyết + 22 phút thực hành	40 phút
5	`HAVING`, `WHERE` vs `HAVING` + **bài 2.4**	9 phút lý thuyết + 32 phút thực hành	41 phút
6	`COUNT(DISTINCT)`, `STRING_AGG` / `GROUP_CONCAT`, `ROUND` / `CAST` + **bài 2.5**	9 phút lý thuyết + 22 phút thực hành	31 phút
7–9	Áp dụng nghiệp vụ, sai lầm thường gặp, tóm tắt, chữa bài, Q&A	Thực hành + thảo luận (làm lại bài khó, đổi chéo chữa bài)	10 phút
	Tổng	~52 phút lý thuyết + demo, ~128 phút thực hành (~71%)	**180 phút**

1. Ôn tập và mở rộng ORDER BY

Ba yêu cầu sắp xếp thường gặp trên danh sách khách hàng, tài khoản. Bấm **Chạy thử** để xem từng câu:

```sql
-- Theo tên A → Z; cùng tên thì theo mã để kết quả ổn định
SELECT customer_id, full_name, city FROM Customers ORDER BY full_name, customer_id;
```

```
8   Bui Thi Lan Ha Noi
29  Bui Van Manh    TP.HCM
19  Cao Van Yen Ha Noi
35  Cong ty CP Chung khoan An Binh  Ha Noi
... (40 dòng)
```

```sql
-- Theo ngày tham gia, mới nhất trước
SELECT customer_id, full_name, created_date
FROM Customers
ORDER BY created_date DESC, customer_id;
```

```
5   Hoang Van Em    2026-09-23
26  Hoang Thi Huong 2026-09-23
12  Ly Thi Quynh    2026-09-19
19  Cao Van Yen 2026-09-14
... (40 dòng)
```

```sql
-- Theo số dư, cao nhất trước
SELECT account_id, account_no, balance FROM Accounts ORDER BY balance DESC, account_id;
```

```
4   ACC00004    8600000000.00
35  ACC00035    8600000000.00
93  ACC00093    8600000000.00
11  ACC00011    3400000000.00
33  ACC00033    3400000000.00
... (93 dòng)
```

Nhắc lại ba điều quan trọng:

- `ORDER BY` **luôn nằm cuối câu lệnh** (trước phần giới hạn số dòng) và là mệnh đề duy nhất được dùng **bí danh cột** trên cả hai hệ.
- Có thể sắp xếp theo **biểu thức** (ví dụ ngay bên dưới).
- **Luôn thêm khóa phụ duy nhất** (thường là khóa chính) khi có khả năng đồng hạng.

Ví dụ sắp xếp theo biểu thức – lệnh có giá trị (khối lượng × giá) lớn nhất trước:

```sql
SELECT order_id, symbol, quantity, price, quantity * price AS gia_tri
FROM Orders
ORDER BY quantity * price DESC, order_id;
```

```
7   VCB 5000    93000.00    465000000.00
104 VCB 5000    93000.00    465000000.00
15  VCS 5000    61000.00    305000000.00
107 VCS 5000    61000.00    305000000.00
... (120 dòng)
```

> **Cảnh báo:** trong dữ liệu mẫu, ba tài khoản `ACC00004`, `ACC00035`, `ACC00093` đều có số dư đúng 8,6 tỷ. Nếu chỉ `ORDER BY balance DESC`, thứ tự giữa ba tài khoản này **không được đảm bảo** và có thể khác nhau giữa SQL Server và MySQL – bài chấm tự động sẽ báo sai dù logic của bạn "đúng".

Thực hành: vừa ôn xong `ORDER BY` với khóa phụ, hãy làm **bài 2.1** để sắp xếp danh sách khách hàng theo tên, theo ngày tham gia và tài khoản theo số dư.

- **Bài 2.1** — Sắp xếp khách hàng theo tên A→Z *(độ khó: dễ)*
- **Bài 2.1** — Sắp xếp khách hàng theo ngày tham gia *(độ khó: dễ)*
- **Bài 2.1** — Sắp xếp tài khoản theo số dư giảm dần *(độ khó: dễ)*

2. Các hàm thống kê tổng hợp cơ bản

Hàm thống kê nhận **nhiều dòng** và trả về **một giá trị**.

Hàm	Ý nghĩa	Ghi chú
`COUNT(*)`	Đếm số dòng	Đếm cả dòng có NULL
`COUNT(cột)`	Đếm dòng mà `cột` **khác NULL**	Bỏ qua NULL
`SUM(cột)`	Tổng	Bỏ qua NULL; không có dòng nào → `NULL`
`AVG(cột)`	Trung bình	Bỏ qua NULL (không tính vào mẫu số)
`MIN(cột)`, `MAX(cột)`	Nhỏ nhất, lớn nhất	Dùng được cho số, chuỗi, ngày

2.1. Ví dụ tổng hợp trên bảng Accounts

```sql
SELECT COUNT(*)     AS so_tai_khoan,
       SUM(balance) AS tong_so_du,
       MIN(balance) AS nho_nhat,
       MAX(balance) AS lon_nhat
FROM Accounts;
```

```
93  66987000000.00  3000000.00  8600000000.00
```

Một câu `SELECT` chỉ dùng hàm thống kê (không có `GROUP BY`) luôn trả về **đúng một dòng**. Kết hợp `WHERE` để thống kê trên một tập con:

```sql
-- Chỉ tính tài khoản đang hoạt động
SELECT COUNT(*) AS so_tai_khoan, SUM(balance) AS tong_so_du
FROM Accounts
WHERE status = 'ACTIVE';
```

```
88  63932000000.00
```

`WHERE` được áp dụng **trước** khi thống kê: chỉ những dòng còn lại mới được đếm/cộng.

2.2. COUNT(*) khác COUNT(cột)

Cột `email` có 5 giá trị `NULL` trong 40 khách hàng:

```sql
SELECT COUNT(*) AS tong_khach_hang, COUNT(email) AS co_email
FROM Customers;
```

```
40  35
```

Đếm số khách hàng đang hoạt động:

```sql
SELECT COUNT(*) AS so_khach_hang_active FROM Customers WHERE status = 'ACTIVE';
```

```
31
```

> **SQL Server vs MySQL:** khi hàm thống kê bỏ qua NULL, SQL Server in thêm thông báo `Warning: Null value is eliminated by an aggregate or other SET operation.` – đó **chỉ là cảnh báo**, kết quả vẫn đúng. Muốn tắt, thêm `SET ANSI_WARNINGS OFF;` ở đầu script. **MySQL** không có cảnh báo này.

2.3. Ba bẫy của SUM/AVG

Bẫy 1 – Tập rỗng. `SUM` trên tập rỗng trả về `NULL` chứ không phải 0, trong khi `COUNT(*)` trả về `0`:

```sql
SELECT COUNT(*) AS so_dong, SUM(balance) AS tong_so_du, COALESCE(SUM(balance), 0) AS
        tong_hoac_0
FROM Accounts
WHERE status = 'KHONG_TON_TAI';
```

```
0   NULL    0.00
```

Bẫy 2 – Trung bình bỏ qua NULL. `AVG(cột)` = tổng ÷ số dòng *có giá trị*, không phải tổng ÷ `COUNT(*)`.

Bẫy 3 – Chia số nguyên (khác biệt lớn giữa hai hệ). T-SQL chia hai số nguyên cho ra số nguyên:

Biểu thức	SQL Server	MySQL
`7 / 2`	`3` (chia nguyên)	`3.5000`
`AVG(quantity)` với `quantity` là INT	`1280` (mất phần lẻ)	`1280.8333`
`AVG(1.0 * quantity)`	`1280.833333`	`1280.8333`

SQL Server:

```sql
SELECT 7 / 2 AS chia_nguyen, 7 / 2.0 AS chia_thap_phan, 7.0 / 2 AS chia_thap_phan_2;
```

```
3   3.500000    3.500000
```

MySQL:

```sql
SELECT 7 / 2 AS chia_thuong, 7 DIV 2 AS chia_nguyen;
```

```
3.5000  3
```

Cùng biểu thức `7 / 2` nhưng SQL Server cho `3`, MySQL cho `3.5000` – đây chính là bẫy khi chuyển câu lệnh giữa hai hệ. MySQL muốn chia nguyên phải dùng toán tử `DIV`.

> **Mẹo:** quy tắc vàng – nếu cần kết quả thập phân, **nhân với `1.0`** (`1.0 * a / b`, `100.0 * a / b`) và làm tròn/ép kiểu rõ ràng (mục 6.3). Cách này chạy giống nhau trên cả hai hệ.

> **Ví dụ nghiệp vụ:** "tài sản trung bình mỗi khách hàng" **không** phải là `AVG(balance)` – đó là số dư trung bình *mỗi tài khoản*. Vì một khách hàng có nhiều tài khoản, phải lấy tổng số dư chia cho số khách hàng **khác nhau**:

```sql
SELECT COUNT(DISTINCT customer_id) AS so_khach_hang,
       SUM(balance)                AS tong_so_du,
       CAST(ROUND(SUM(balance) / COUNT(DISTINCT customer_id), 2) AS DECIMAL(18,2)) AS
        tb_moi_khach,
       CAST(ROUND(AVG(balance), 2) AS DECIMAL(18,2))                               AS
        tb_moi_tai_khoan
FROM Accounts
WHERE status = 'ACTIVE';
```

```
35  63932000000.00  1826628571.43   726500000.00
```

Nhìn kết quả: trung bình mỗi khách hàng nắm ~1,83 tỷ, mỗi tài khoản trung bình ~0,73 tỷ – hai con số trả lời hai câu hỏi khác nhau.

`MIN` / `MAX` dùng được cho ngày và chuỗi – ví dụ ngày mở hồ sơ sớm nhất và muộn nhất:

```sql
SELECT MIN(created_date) AS som_nhat, MAX(created_date) AS muon_nhat FROM Customers;
```

```
2018-06-18  2026-09-23
```

Thực hành: vừa học xong `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, hãy làm **bài 2.2** để thống kê tài sản trên tài khoản đang hoạt động, tính tài sản trung bình mỗi khách hàng và đếm khách hàng đang hoạt động.

- **Bài 2.2** — Thống kê tài sản trên tài khoản đang hoạt động *(độ khó: dễ)*
- **Bài 2.2** — Tài sản trung bình mỗi khách hàng *(độ khó: trung bình)*
- **Bài 2.2** — Số khách hàng đang hoạt động *(độ khó: dễ)*

3. Mệnh đề giới hạn số dòng: TOP, LIMIT và OFFSET

Khi chỉ cần "N dòng đầu" (top 10, 5 lệnh lớn nhất...) hoặc lấy dữ liệu theo từng trang, chúng ta phải **giới hạn** kết quả *sau khi đã sắp xếp*. Đây là chỗ **hai hệ khác cú pháp**.

3.1. Lấy N hàng đầu tiên

SQL Server – `TOP` ngay sau `SELECT`:

```
SELECT TOP 5 customer_id, total_trade_value, trade_count
FROM CustomerStats
ORDER BY total_trade_value DESC, customer_id;
```

MySQL – `LIMIT` ở cuối câu lệnh:

```
SELECT customer_id, total_trade_value, trade_count
FROM CustomerStats
ORDER BY total_trade_value DESC, customer_id
LIMIT 5;
```

Cả hai cùng cho năm khách hàng có tổng giá trị giao dịch lớn nhất:

```
16  703300000.00    6
36  422600000.00    4
22  420525000.00    7
35  256600000.00    4
5   220400000.00    5
```

Chuẩn ANSI – dùng được ở SQL Server 2012+ (không có ở MySQL): `OFFSET 0 ROWS FETCH NEXT 5 ROWS ONLY`.

	SQL Server	MySQL
Lấy n dòng đầu	`SELECT TOP n ...` (ngay sau `SELECT`)	`... LIMIT n` (cuối câu lệnh)
Bỏ qua k dòng	`OFFSET k ROWS`	`OFFSET k`
Đồng hạng	`TOP n WITH TIES`	không có

Lưu ý quan trọng:

- Luôn có `ORDER BY` khi dùng `TOP` / `LIMIT`; không có nó, "5 dòng đầu" là 5 dòng bất kỳ.
- **Đồng hạng:** ba tài khoản 8,6 tỷ đang cùng đứng đầu. `SELECT TOP 2 WITH TIES ... ORDER BY balance DESC` (chỉ SQL Server) trả về **3 dòng** (lấy thêm mọi dòng bằng dòng thứ 2), còn `TOP 2` / `LIMIT 2` cắt cứng đúng 2 dòng và nếu thiếu khóa phụ thì việc 2 trong 3 tài khoản nào được chọn là **không xác định**.

> **Cảnh báo:** bài tập của khóa **không dùng** `WITH TIES` mà luôn thêm khóa phụ để top-N tất định trên cả hai hệ. `TOP` không kèm `ORDER BY` là lỗi logic phổ biến nhất khi làm báo cáo "top N".

3.2. Phân trang bằng OFFSET

Một màn hình quản trị không hiện 40, hay 40.000 khách hàng cùng lúc mà chia thành từng **trang**. Công thức: **bỏ qua `(số_trang − 1) × kích_thước_trang` dòng rồi lấy `kích_thước_trang` dòng.**

SQL Server – `OFFSET ... FETCH` (bắt buộc đi cùng `ORDER BY`, đặt sau `ORDER BY`):

```
-- Trang 2, mỗi trang 10 khách hàng
SELECT customer_id, full_name
FROM Customers
ORDER BY customer_id
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

MySQL:

```
-- Trang 2, mỗi trang 10 khách hàng
SELECT customer_id, full_name
FROM Customers
ORDER BY customer_id
LIMIT 10 OFFSET 10;      -- hoặc: LIMIT 10, 10  (cú pháp LIMIT offset, số_dòng)
```

Cùng cho mười khách hàng có `customer_id` 11–20:

```
11  Duong Van Phong
12  Ly Thi Quynh
13  Trinh Van Son
14  Phan Thi Thao
... (10 dòng)
```

Trang (10 dòng/trang)	Bỏ qua	SQL Server	MySQL
1	0	`OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY`	`LIMIT 10 OFFSET 0`
2	10	`OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY`	`LIMIT 10 OFFSET 10`
3	20	`OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY`	`LIMIT 10 OFFSET 20`
n, kích thước k	(n−1)·k	`OFFSET (n−1)*k ROWS FETCH NEXT k ROWS ONLY`	`LIMIT k OFFSET (n−1)*k`

Sai lầm thường gặp khi phân trang:

Sai lầm	Hậu quả	Cách đúng
`ORDER BY created_date DESC` nhưng nhiều khách hàng cùng ngày	Một dòng có thể xuất hiện ở cả hai trang hoặc biến mất	Thêm khóa duy nhất: `ORDER BY created_date DESC, customer_id`
Không xử lý trang cuối chưa đầy	Với 40 khách hàng và 15 dòng/trang, trang 3 chỉ có 10 dòng, trang 4 rỗng	Ứng dụng phải xử lý kết quả ít dòng hoặc rỗng
`OFFSET` rất lớn trên bảng hàng triệu dòng	Chậm (vẫn phải duyệt rồi bỏ qua các dòng đầu)	Kỹ thuật "keyset": `WHERE id > id_cuối_trang_trước`

> **Lưu ý:** bài **2.3** gồm cả **phân trang** (mục này) lẫn **Top 10 khách hàng giàu nhất** (cần `GROUP BY`, mục 4.5). Chúng ta sẽ làm bài 2.3 ngay sau khi học xong `GROUP BY` để làm trọn cả hai.

4. Mệnh đề GROUP BY

Hàm thống kê ở mục 2 gom **cả bảng** thành một dòng. Thực tế ta thường muốn thống kê **theo từng nhóm**: mỗi thành phố, mỗi loại tài khoản, mỗi mã cổ phiếu. `GROUP BY` chia các dòng thành các nhóm có cùng giá trị ở cột nhóm, rồi hàm thống kê được tính **riêng trong từng nhóm**.

Hình dung với 6 tài khoản của hai khách hàng 2 và 3, khi nhóm theo `customer_id` và tính `SUM(balance)`:

Bước 1 – dòng gốc			Bước 2 – chia nhóm theo `customer_id`	Bước 3 – `SUM(balance)`
ACC00002	customer 2	8.000.000	Nhóm **customer 2** (3 dòng)	**8.640.000.000**
ACC00003	customer 2	32.000.000	↑	↑
ACC00004	customer 2	8.600.000.000	↑	↑
ACC00005	customer 3	3.000.000	Nhóm **customer 3** (3 dòng)	**168.000.000**
ACC00006	customer 3	67.000.000	↑	↑
ACC00007	customer 3	98.000.000	↑	↑

Kết quả cuối cùng chỉ còn **một dòng cho mỗi nhóm**.

Hình dưới cho thấy `GROUP BY` làm gì với dữ liệu: các dòng cùng giá trị cột nhóm được gom lại, hàm tổng hợp tính trên từng nhóm, cuối cùng `HAVING` giữ hoặc loại cả nhóm.

Hình minh họa: Luồng chạy của GROUP BY (xem hình động trên trang bài giảng trực tuyến).

4.1. Nhóm theo một cột

```sql
SELECT city, COUNT(*) AS so_khach_hang
FROM Customers
GROUP BY city
ORDER BY so_khach_hang DESC, city;
```

```
Ha Noi  22
TP.HCM  7
Da Nang 5
Can Tho 3
Hai Phong   3
```

Đọc câu lệnh: "chia khách hàng theo `city`, với mỗi thành phố đếm số dòng, rồi sắp xếp".

4.2. Nhóm theo nhiều cột

Mỗi **tổ hợp** giá trị của các cột nhóm tạo thành một nhóm:

```sql
SELECT city, customer_type, COUNT(*) AS so_khach_hang
FROM Customers
GROUP BY city, customer_type
ORDER BY city, customer_type;
```

```
Can Tho INDIVIDUAL  3
Da Nang INDIVIDUAL  5
Ha Noi  INDIVIDUAL  16
Ha Noi  INSTITUTION 6
Hai Phong   INDIVIDUAL  3
TP.HCM  INDIVIDUAL  3
TP.HCM  INSTITUTION 4
```

4.3. Kết hợp GROUP BY với nhiều hàm thống kê

```sql
SELECT account_type,
       COUNT(*)     AS so_tai_khoan,
       SUM(balance) AS tong_so_du,
       MAX(balance) AS lon_nhat
FROM Accounts
WHERE status = 'ACTIVE'
GROUP BY account_type
ORDER BY account_type;
```

```
CASH    53  45560000000.00  8600000000.00
MARGIN  35  18372000000.00  3400000000.00
```

4.4. Lưu ý khi dùng GROUP BY

Buổi 1 đã giới thiệu **thứ tự xử lý logic** của câu `SELECT` (hình ở mục 4.3 Buổi 1). Giờ hai bước còn thiếu trong hình đã có vai trò:

Bước	Mệnh đề	Việc làm
1	`FROM`	Lấy bảng nguồn
2	`WHERE`	Lọc **dòng**
3	`GROUP BY`	Chia dòng còn lại thành nhóm
4	`HAVING`	Lọc **nhóm** (học ở mục 5)
5	`SELECT`	Tính hàm thống kê, chọn cột, đặt bí danh
6	`ORDER BY`	Sắp xếp kết quả
7	`TOP` / `LIMIT`	Cắt bớt dòng

Từ thứ tự này suy ra các quy tắc sau:

1. **Quy tắc cột trong SELECT:** mọi cột trong `SELECT` phải là (a) cột có trong `GROUP BY` hoặc (b) nằm trong một hàm thống kê. Câu `SELECT customer_id, city FROM Customers GROUP BY city` **báo lỗi** trên cả SQL Server lẫn MySQL 8 (`ONLY_FULL_GROUP_BY`): mỗi thành phố có nhiều khách hàng, nên `customer_id` nào?
2. **NULL cũng là một nhóm:** `GROUP BY referrer_id` cho một nhóm `NULL` gồm 32 khách hàng chưa có người giới thiệu (ví dụ ngay bên dưới).
3. `WHERE` lọc *dòng* trước khi nhóm; điều kiện trên kết quả thống kê phải để sau (mục 5).
4. **Bí danh:** `ORDER BY so_khach_hang` được cả hai hệ; nhưng `GROUP BY` bằng bí danh chỉ MySQL chấp nhận – hãy viết lại **biểu thức/cột gốc** trong `GROUP BY` để chạy được ở cả hai.
5. **`GROUP BY` không tự sắp xếp kết quả** – luôn thêm `ORDER BY` nếu cần thứ tự.

Ví dụ nhóm `NULL` :

```sql
SELECT referrer_id, COUNT(*) AS so_khach_hang
FROM Customers
GROUP BY referrer_id
ORDER BY referrer_id;
```

```
NULL    32
1   2
2   1
3   2
8   2
31  1
```

> **Cảnh báo:** đừng "chữa cháy" lỗi `GROUP BY` bằng cách thêm mọi cột vào `GROUP BY` – bạn sẽ chia nhóm mịn đến mức mỗi nhóm chỉ có 1 dòng và số liệu mất ý nghĩa.

4.5. Ứng dụng: Top 10 khách hàng có tài sản lớn nhất

"Tài sản" của một khách hàng = tổng số dư các tài khoản đang hoạt động. Chưa có `JOIN`, ta nhóm theo `customer_id` trên bảng `Accounts`, sắp xếp giảm dần rồi lấy 10 dòng đầu.

SQL Server:

```sql
SELECT TOP 10 customer_id, SUM(balance) AS tong_tai_san
FROM Accounts
WHERE status = 'ACTIVE'
GROUP BY customer_id
ORDER BY tong_tai_san DESC, customer_id;
```

MySQL:

```sql
SELECT customer_id, SUM(balance) AS tong_tai_san
FROM Accounts
WHERE status = 'ACTIVE'
GROUP BY customer_id
ORDER BY tong_tai_san DESC, customer_id
LIMIT 10;
```

Cùng cho kết quả:

```
17  12067000000.00
40  9500000000.00
2   8640000000.00
34  4690000000.00
38  4661000000.00
32  4345000000.00
5   3432000000.00
31  1735000000.00
6   1600000000.00
30  1402000000.00
```

Khóa phụ `customer_id` bảo đảm nếu hai khách hàng cùng tổng tài sản thì luôn xếp theo mã. (Ở Buổi 8 ta sẽ ghép thêm bảng `Customers` để hiện tên khách hàng.)

Thực hành: vừa học xong `TOP` / `LIMIT`, phân trang và `GROUP BY`, hãy làm **bài 2.3** – lấy Top 10 khách hàng có tài sản lớn nhất và phân trang danh sách khách hàng.

- **Bài 2.3** — Top 10 khách hàng có tài sản lớn nhất *(độ khó: trung bình)*
- **Bài 2.3** — Phân trang danh sách khách hàng *(độ khó: trung bình)*

5. Mệnh đề HAVING

Sau khi nhóm, ta thường muốn **lọc theo kết quả thống kê của nhóm**: "chỉ giữ thành phố có ít nhất 5 khách hàng", "khách hàng có tổng tài sản trên 4 tỷ". `WHERE` **không** làm được việc này vì `WHERE` chạy trước khi nhóm, chưa có `COUNT` / `SUM`.

	`WHERE`	`HAVING`
Lọc cái gì	từng **dòng**	từng **nhóm**
Chạy khi nào	trước `GROUP BY`	sau `GROUP BY`
Dùng được hàm thống kê	Không	Có
Dùng được cột thường	Có	Chỉ cột trong `GROUP BY` hoặc hàm thống kê

```sql
-- Thành phố có ít nhất 5 khách hàng
SELECT city, COUNT(*) AS so_khach_hang
FROM Customers
GROUP BY city
HAVING COUNT(*) >= 5
ORDER BY so_khach_hang DESC;
```

```
Ha Noi  22
TP.HCM  7
Da Nang 5
```

Ứng dụng thực tế: WHERE + HAVING cùng lúc

"Khách hàng có tổng tài sản (tài khoản ACTIVE) trên 4 tỷ":

```sql
SELECT customer_id, COUNT(*) AS so_tai_khoan, SUM(balance) AS tong_so_du
FROM Accounts
WHERE status = 'ACTIVE'          -- lọc dòng trước: bỏ tài khoản đã đóng
GROUP BY customer_id
HAVING SUM(balance) > 4000000000 -- lọc nhóm sau: giữ khách hàng lớn
ORDER BY tong_so_du DESC, customer_id;
```

```
17  3   12067000000.00
40  3   9500000000.00
2   3   8640000000.00
34  4   4690000000.00
38  4   4661000000.00
32  4   4345000000.00
```

> **Mẹo:** điều kiện chỉ dùng cột của từng dòng (trạng thái, loại, ngày...) → đặt ở `WHERE` (nhanh hơn vì loại bỏ dòng sớm); điều kiện dùng kết quả `COUNT/SUM/AVG/MIN/MAX` → `HAVING`.

Sai lầm thường gặp:

Sai lầm	Hậu quả	Cách đúng
`HAVING status = 'ACTIVE'` khi `status` không có trong `GROUP BY`	Lỗi	Đưa điều kiện dòng lên `WHERE`
Dùng bí danh trong `HAVING` (`HAVING tong > 4000000000`)	MySQL chấp nhận, **SQL Server báo `Invalid column name`**	Lặp lại biểu thức: `HAVING SUM(balance) > 4000000000`
Dùng hàm thống kê trong `WHERE` (`WHERE COUNT(*) > 3`)	Lỗi trên cả hai hệ	Chuyển sang `HAVING`

Thực hành: vừa học xong `GROUP BY` nhiều cột và `HAVING`, hãy làm **bài 2.4** – thống kê khách hàng theo loại/trạng thái, lọc khách hàng nhiều tài khoản tài sản lớn và làm báo cáo giá trị khớp lệnh theo mã. Bài 2.4c (hard) còn dùng `COUNT(DISTINCT)` và `ROUND` / `CAST` – bạn đã gặp ở mục 2.3, và sẽ học kỹ hơn ở mục 6.

- **Bài 2.4** — Thống kê khách hàng theo loại và trạng thái *(độ khó: dễ)*
- **Bài 2.4** — Khách hàng có nhiều tài khoản và tài sản lớn *(độ khó: trung bình)*
- **Bài 2.4** — Báo cáo giá trị khớp lệnh theo mã *(độ khó: khó)*

6. Các hàm tổng hợp nâng cao

6.1. COUNT(DISTINCT ...) – đếm giá trị duy nhất

`COUNT(cột)` đếm số dòng; `COUNT(DISTINCT cột)` đếm số **giá trị khác nhau**.

```sql
SELECT COUNT(*)                  AS so_lenh,
       COUNT(DISTINCT account_id) AS so_tai_khoan_da_dat_lenh,
       COUNT(DISTINCT symbol)     AS so_ma_duoc_giao_dich
FROM Orders;
```

```
120 59  12
```

120 lệnh do 59 tài khoản khác nhau đặt, trên đủ 12 mã. Thường dùng theo nhóm, ví dụ trên từng trạng thái lệnh:

```sql
SELECT status, COUNT(*) AS so_lenh,
       COUNT(DISTINCT account_id) AS so_tai_khoan,
       COUNT(DISTINCT symbol)     AS so_ma
FROM Orders
GROUP BY status
ORDER BY status;
```

```
CANCELLED   17  14  9
MATCHED 69  42  12
PARTIAL 17  16  10
PENDING 17  14  9
```

> **Lưu ý:** `COUNT(DISTINCT ...)` bỏ qua NULL và là phép tính "đắt" trên bảng lớn (phải loại trùng).

6.2. Nối chuỗi trong nhóm: STRING_AGG / GROUP_CONCAT

Đôi khi ta muốn gộp các giá trị của một nhóm thành **một chuỗi** (ví dụ mỗi sàn có những mã nào). Hai hệ dùng hàm khác tên:

SQL Server (2017+) – `STRING_AGG(cột, dấu_phân_cách)`, sắp xếp phần tử bằng `WITHIN GROUP (ORDER BY ...)`:

```sql
SELECT exchange, COUNT(*) AS so_ma,
       STRING_AGG(symbol, ',') WITHIN GROUP (ORDER BY symbol) AS danh_sach_ma
FROM Securities
GROUP BY exchange
ORDER BY exchange;
```

MySQL – `GROUP_CONCAT(cột ORDER BY ... SEPARATOR 'dấu')`:

```sql
SELECT exchange, COUNT(*) AS so_ma,
       GROUP_CONCAT(symbol ORDER BY symbol SEPARATOR ',') AS danh_sach_ma
FROM Securities
GROUP BY exchange
ORDER BY exchange;
```

Cùng kết quả:

```
HNX 8   CEO,IDC,MBS,NVB,PVI,PVS,SHS,VCS
HOSE    4   FPT,HPG,VCB,VNM
```

Một ví dụ khác – liệt kê số tài khoản của từng khách hàng:

SQL Server:

```sql
SELECT customer_id, STRING_AGG(account_no, ', ') WITHIN GROUP (ORDER BY account_no) AS
        cac_tai_khoan
FROM Accounts
WHERE customer_id IN (2, 3, 5)
GROUP BY customer_id
ORDER BY customer_id;
```

MySQL:

```sql
SELECT customer_id, GROUP_CONCAT(account_no ORDER BY account_no SEPARATOR ', ') AS
        cac_tai_khoan
FROM Accounts
WHERE customer_id IN (2, 3, 5)
GROUP BY customer_id
ORDER BY customer_id;
```

```
2   ACC00002, ACC00003, ACC00004
3   ACC00005, ACC00006, ACC00007
5   ACC00010, ACC00011
```

> **Cảnh báo:** nếu **không** chỉ định thứ tự (`WITHIN GROUP` / `ORDER BY` trong hàm), thứ tự phần tử trong chuỗi không được đảm bảo và có thể khác nhau giữa hai hệ → luôn chỉ định để kết quả tất định. `NULL` bị bỏ qua khi nối. MySQL giới hạn độ dài mặc định của `GROUP_CONCAT` (1024 ký tự) – đủ cho các bài của khóa.

Thực hành: vừa học xong `COUNT(DISTINCT)` và `STRING_AGG` / `GROUP_CONCAT`, hãy làm **bài 2.5** – thống kê lệnh theo trạng thái và gộp danh sách mã chứng khoán theo sàn.

- **Bài 2.5** — Thống kê lệnh theo trạng thái *(độ khó: trung bình)*
- **Bài 2.5** — Danh sách mã chứng khoán theo sàn *(độ khó: trung bình)*

6.3. Định dạng số trong thống kê: ROUND và CAST

- **`ROUND(x, n)`** làm tròn `x` đến `n` chữ số thập phân; `n` âm làm tròn về hàng chục/trăm/nghìn (`ROUND(1250000, -3)` → làm tròn đến nghìn).
- **`CAST(biểu_thức AS kiểu)`** ép kiểu, ví dụ `CAST(x AS DECIMAL(18,2))` – tổng 18 chữ số, trong đó 2 chữ số thập phân.

Vì sao cần cả hai? Hai hệ **in** kết quả `ROUND` khác nhau: với `ROUND(1280.8333..., 2)`, SQL Server in `1280.830000` (giữ số chữ số thập phân của kiểu gốc) còn MySQL in `1280.83`. Bọc thêm `CAST(... AS DECIMAL(18,2))` để cả hai in đúng hai chữ số:

```sql
SELECT CAST(ROUND(AVG(1.0 * quantity), 2) AS DECIMAL(18,2)) AS khoi_luong_tb
FROM Orders;
```

```
1280.83
```

Ví dụ báo cáo thị trường – thống kê giá của từng mã trong 10 phiên gần nhất:

```sql
SELECT symbol,
       MAX(high_price) AS cao_nhat,
       MIN(low_price)  AS thap_nhat,
       CAST(ROUND(AVG(close_price), 2) AS DECIMAL(18,2)) AS dong_cua_tb,
       SUM(volume)     AS tong_khoi_luong
FROM MarketPrices
GROUP BY symbol
ORDER BY symbol;
```

```
CEO 22110.00    19030.00    20306.00    43500000
FPT 126600.00   115800.00   121440.00   53700000
HPG 27675.00    26055.00    27027.00    41800000
IDC 54750.00    49750.00    51950.00    54700000
... (12 dòng)
```

Đổi đơn vị sang tỷ đồng: `CAST(ROUND(SUM(balance) / 1000000000.0, 2) AS DECIMAL(18,2))` (chia cho `1000000000.0` – có `.0` để không bị chia nguyên):

```sql
SELECT CAST(ROUND(SUM(balance) / 1000000000.0, 2) AS DECIMAL(18,2)) AS tong_so_du_ty
FROM Accounts
WHERE status = 'ACTIVE';
```

```
63.93
```

> **SQL Server vs MySQL:** `CAST(3.5 AS INT)` cho `3` (cắt phần lẻ) trong SQL Server, còn MySQL không có kiểu `INT` trong `CAST` mà dùng `CAST(3.5 AS SIGNED)` và cho `4` (làm tròn). Để tránh sai khác, hãy dùng `ROUND` rõ ràng rồi mới ép kiểu, hoặc ép sang `DECIMAL` như trên.

SQL Server:

```sql
SELECT CAST(3.5 AS INT) AS ep_kieu_int;
```

```
3
```

MySQL:

```sql
SELECT CAST(3.5 AS SIGNED) AS ep_kieu_signed;
```

```
4
```

7. Áp dụng: từ câu hỏi nghiệp vụ đến báo cáo

Yêu cầu: *"Với các lệnh đã khớp hết, cho biết mỗi mã có bao nhiêu lệnh, tổng giá trị (khối lượng × giá), và chỉ giữ những mã có từ 5 lệnh trở lên; mã giá trị lớn nhất lên đầu."*

Bóc tách: bảng `Orders`, `WHERE status = 'MATCHED'`; nhóm theo `symbol`; thống kê `COUNT(*)`, `SUM(quantity * price)`; lọc nhóm bằng `HAVING COUNT(*) >= 5`; sắp `ORDER BY` giá trị giảm dần, thêm `symbol` làm khóa phụ.

```sql
SELECT symbol, COUNT(*) AS so_lenh, SUM(quantity * price) AS gia_tri
FROM Orders
WHERE status = 'MATCHED'
GROUP BY symbol
HAVING COUNT(*) >= 5
ORDER BY gia_tri DESC, symbol;
```

```
VCS 6   803950000.00
VCB 8   563650000.00
VNM 9   514050000.00
PVS 8   383450000.00
MBS 6   202850000.00
... (9 dòng)
```

Trước khi nộp bài, hãy tự kiểm tra ba điều: (1) cột trong `SELECT` đã nằm trong `GROUP BY` hoặc hàm thống kê chưa? (2) điều kiện đặt đúng chỗ `WHERE` / `HAVING` chưa? (3) `ORDER BY` đã đủ khóa phụ để kết quả không phụ thuộc thứ tự vật lý chưa?

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Nhầm `AVG(balance)` với "trung bình mỗi khách hàng"	Số trung bình mỗi *tài khoản* – sai đơn vị	`SUM(balance) / COUNT(DISTINCT customer_id)`
Chia số nguyên trên SQL Server (`AVG` của INT, `a / b`)	Mất phần thập phân, hai hệ cho kết quả khác nhau	Nhân `1.0` , bọc `CAST(ROUND(..., 2) AS DECIMAL(18,2))`
Cột không nằm trong `GROUP BY` cũng không trong hàm thống kê	Lỗi (`ONLY_FULL_GROUP_BY`)	Thêm vào `GROUP BY` nếu cần chia nhóm mịn hơn, hoặc bọc bằng hàm thống kê
Lọc theo hàm thống kê trong `WHERE` , hoặc lọc điều kiện dòng trong `HAVING`	Lỗi hoặc chậm	Điều kiện dòng → `WHERE` ; điều kiện nhóm → `HAVING`
`TOP` / `LIMIT` /phân trang không có `ORDER BY` hoặc thiếu khóa phụ	Dòng lặp/thiếu giữa các trang, chấm sai	`ORDER BY` luôn có khóa duy nhất ở cuối
`STRING_AGG` / `GROUP_CONCAT` không chỉ định thứ tự	Chuỗi khác nhau giữa hai hệ	`WITHIN GROUP (ORDER BY ...)` / `ORDER BY ... SEPARATOR`
Quên `SUM` trên tập rỗng cho `NULL`	Báo cáo hiện `NULL` thay vì 0	`COALESCE(SUM(...), 0)`

Tóm tắt

Nội dung	Ghi nhớ
Hàm thống kê	`COUNT` , `SUM` , `AVG` , `MIN` , `MAX` ; bỏ qua NULL (trừ `COUNT(*)`); `SUM` trên tập rỗng là NULL
`COUNT(*)` vs `COUNT(cột)`	Đếm mọi dòng vs đếm dòng khác NULL
Chia số nguyên	SQL Server chia nguyên (`AVG` của INT, `7/2`); nhân `1.0` để có phần thập phân
Top-N	SQL Server `SELECT TOP n ...` ; MySQL `... LIMIT n` ; luôn có `ORDER BY` + khóa phụ
Phân trang	SQL Server `OFFSET (n−1)∗k ROWS FETCH NEXT k ROWS ONLY` ; MySQL `LIMIT k OFFSET (n−1)∗k`
`GROUP BY`	Chia nhóm; cột trong `SELECT` phải thuộc `GROUP BY` hoặc hàm thống kê
`WHERE` vs `HAVING`	Lọc dòng (trước nhóm) vs lọc nhóm (sau nhóm, dùng được hàm thống kê)
`COUNT(DISTINCT x)`	Đếm giá trị khác nhau
Nối chuỗi trong nhóm	SQL Server `STRING_AGG(x, ',') WITHIN GROUP (ORDER BY x)` ; MySQL `GROUP_CONCAT(x ORDER BY x SEPARATOR ',')`
Định dạng số	`CAST(ROUND(x, 2) AS DECIMAL(18,2))` để hai hệ in giống nhau

Gợi ý mở rộng

- Thử tìm mã có **khối lượng khớp trung bình mỗi lệnh** cao nhất: `GROUP BY symbol` trên `Orders` (chú ý chia số nguyên) rồi `TOP 1` / `LIMIT 1`.
- Thống kê **theo tháng**: `GROUP BY YEAR(order_date), MONTH(order_date)` — `YEAR` / `MONTH` giống nhau ở hai hệ (Buổi 4 sẽ học kỹ các hàm ngày).
- Trong bảng `CustomerStats`, tính giá trị giao dịch trung bình mỗi lần giao dịch (`total_trade_value / trade_count`) cho các khách hàng có `trade_count > 0` – chú ý tránh chia cho 0.
- Đọc thêm: `GROUP BY ... WITH ROLLUP` (MySQL) và `GROUP BY ROLLUP(...)` (SQL Server) để có dòng tổng cộng trong báo cáo; hàm cửa sổ (Buổi 3) sẽ giải quyết những bài "top N *trong mỗi nhóm*".

Bài tập thực hành của buổi

Danh sách đầy đủ các bài của buổi (bạn đã gặp từng nhóm ngay trong bài giảng), nộp trên Codelab bằng SQL Server hoặc MySQL:

- **Bài 2.1** — Sắp xếp khách hàng theo tên A→Z *(độ khó: dễ)*
- **Bài 2.1** — Sắp xếp khách hàng theo ngày tham gia *(độ khó: dễ)*
- **Bài 2.1** — Sắp xếp tài khoản theo số dư giảm dần *(độ khó: dễ)*
- **Bài 2.2** — Thống kê tài sản trên tài khoản đang hoạt động *(độ khó: dễ)*
- **Bài 2.2** — Tài sản trung bình mỗi khách hàng *(độ khó: trung bình)*
- **Bài 2.2** — Số khách hàng đang hoạt động *(độ khó: dễ)*
- **Bài 2.3** — Top 10 khách hàng có tài sản lớn nhất *(độ khó: trung bình)*
- **Bài 2.3** — Phân trang danh sách khách hàng *(độ khó: trung bình)*
- **Bài 2.4** — Thống kê khách hàng theo loại và trạng thái *(độ khó: dễ)*
- **Bài 2.4** — Khách hàng có nhiều tài khoản và tài sản lớn *(độ khó: trung bình)*
- **Bài 2.5** — Thống kê lệnh theo trạng thái *(độ khó: trung bình)*
- **Bài 2.5** — Danh sách mã chứng khoán theo sàn *(độ khó: trung bình)*
- **Bài 2.4** — Báo cáo giá trị khớp lệnh theo mã *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: xem, đếm và xuất kết quả báo cáo

Làm trên SSMS (đã cài ở Buổi 1), sau khi nạp `hnx_securities`. Buổi này bạn làm quen với **cách đọc và xuất kết quả**, kỹ năng cần ngay khi làm báo cáo thống kê.

1. Chọn nơi hiển thị kết quả. Menu **Query** → **Results To**: *Results to Grid* (**Ctrl+D**, lưới, mặc định), *Results to Text* (**Ctrl+T**, văn bản thuần, dễ xem cột thẳng hàng) hoặc *Results to File* (**Ctrl+Shift+F**, ghi ra file `.rpt`). Nhấn **Ctrl+R** để ẩn/hiện khung kết quả khi cần thêm chỗ soạn câu lệnh.

2. Đọc thanh trạng thái. Góc dưới cửa sổ truy vấn có số dòng trả về (ví dụ *40 rows*) và thời gian chạy. Hãy đối chiếu số dòng này với kỳ vọng trước khi tin vào kết quả.

3. Thử `COUNT(*)` và `COUNT(cột)` ngay trên máy.

```sql
USE hnx_securities;
SELECT COUNT(*)     AS so_dong,
       COUNT(email) AS so_email_khac_null,
       COUNT(DISTINCT city) AS so_thanh_pho
FROM Customers;
```

Đối chiếu: `so_dong` lớn hơn `so_email_khac_null` khi có khách chưa có email (NULL không được đếm).

4. Thử phân trang với `OFFSET ... FETCH` (trang 2, mỗi trang 10 khách hàng có giá trị giao dịch lớn nhất):

```sql
SELECT customer_id, total_trade_value
FROM CustomerStats
ORDER BY total_trade_value DESC, customer_id
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

5. Xuất kết quả sang Excel.

1. Chạy một truy vấn `GROUP BY`, ví dụ số tài khoản và tổng số dư theo loại tài khoản.
2. Chuột phải vào lưới kết quả → **Save Results As...** → chọn đuôi **CSV**, đặt tên, lưu.
3. Mở file bằng Excel (nếu tiếng Việt lỗi font: **Data** → **From Text/CSV**, chọn *File Origin* là **65001: Unicode (UTF-8)**).
4. Nếu cần tên cột khi sao chép: chuột phải → **Copy with Headers**, rồi dán vào Excel.

> **Mẹo:** muốn file CSV luôn có tên cột, vào **Tools** → **Options** → **Query Results** → **SQL Server** → **Results to Grid**, tick **Include column headers when copying or saving the results**.

Với MySQL Workbench: kết quả hiện ở lưới *Result Grid*; dùng biểu tượng **Export recordset to an external file** phía trên lưới để lưu CSV; `LIMIT 10 OFFSET 10` thay cho `OFFSET ... FETCH`.

Tự kiểm tra: làm lại **bài 2.3** trên SSMS, kiểm tra số dòng ở thanh trạng thái, rồi mới nộp trên Codelab.

Buổi 3 — Window Functions (hàm cửa sổ)

Về mục lục

Đến hết buổi 2, các anh chị đã quen với `GROUP BY`: gom nhiều dòng thành một dòng để tính tổng, trung bình, đếm. Trong công việc ở công ty chứng khoán, rất nhiều báo cáo lại đòi hỏi điều ngược lại: **giữ nguyên từng dòng** nhưng đặt mỗi dòng cạnh một con số của "cả nhóm" — ví dụ giá trị giao dịch của mã VCB **và** tỷ trọng của nó trong cả thị trường, giá đóng cửa hôm nay **và** giá phiên trước, thứ hạng của mã trong ngày, đường trung bình động 3 phiên... Đó chính là đất dụng võ của **window functions**.

Buổi này dùng chủ yếu bảng `MarketPrices` (12 mã × 10 phiên từ 2026-09-14 đến 2026-09-25) và một chút bảng `Trades`. Cấu trúc các bảng xem lại ở mục **"Bộ dữ liệu mẫu của khóa học"** trong Buổi 1. "Hôm nay" của bộ dữ liệu luôn là **2026-09-25**.

> **Lưu ý:** Yêu cầu phiên bản: **SQL Server 2012 trở lên** và **MySQL 8.0 trở lên** có đầy đủ hàm cửa sổ. MySQL 5.7 trở về trước **không** hỗ trợ.

Mục tiêu buổi học

- Sau buổi này bạn có thể giải thích khái niệm hàm cửa sổ, viết đúng cú pháp `OVER (PARTITION BY ... ORDER BY ...)` và phân biệt với `GROUP BY`.
- Sau buổi này bạn có thể dùng `SUM/AVG/MIN/MAX ... OVER` để so sánh từng dòng với tổng thể mà không gộp dòng.
- Sau buổi này bạn có thể xếp hạng bằng `ROW_NUMBER`, `RANK`, `DENSE_RANK`, xử lý đồng hạng và lấy Top N theo nhóm.
- Sau buổi này bạn có thể phân tích biến động giá bằng `LAG` / `LEAD`: mức tăng/giảm và % biến động so với phiên trước.
- Sau buổi này bạn có thể dùng khung cửa sổ `ROWS BETWEEN ...` để tính giá trị lũy kế và đường trung bình động MA3/MA5.

Bản đồ buổi học

Buổi học 3 giờ (180 phút), khoảng 29% lý thuyết + demo + Q&A (~52 phút) và 71% thực hành (~128 phút). Mỗi phần lý thuyết được **xen kẽ ngay** bằng bài thực hành tương ứng: học đến đâu làm đến đó.

Phần	Nội dung	Thời lượng gợi ý
1	Tổng quan Window Functions, so sánh với `GROUP BY`	10 phút (lý thuyết + demo)
2	Window Aggregate: `SUM/AVG/MIN/MAX/COUNT ... OVER`	8 phút (lý thuyết + demo)
3	Ranking: `ROW_NUMBER`, `RANK`, `DENSE_RANK`, đồng hạng, Top N	10 phút (lý thuyết + demo)
Thực hành	**Bài 3.1** — Xếp hạng và phân loại cổ phiếu	44 phút
4	`LAG` và `LEAD` — biến động giá	9 phút (lý thuyết + demo)
Thực hành	**Bài 3.2** — Chuỗi thời gian và biến động giá	44 phút
5	Window Frame và Moving Average	9 phút (lý thuyết + demo)
Thực hành	**Bài 3.3** — Giá trị lũy kế và đường trung bình động	40 phút
6	Tổng kết và Q&A	6 phút (thảo luận)
	Tổng	**180 phút**: lý thuyết + demo + Q&A ~52 phút, thực hành ~128 phút (~71%)

---

1. Tổng quan Window Functions

1.1. Khái niệm

Một **hàm cửa sổ** tính toán trên một *tập các dòng liên quan đến dòng hiện tại* (gọi là "cửa sổ"), nhưng **không gộp các dòng lại**. Mỗi dòng đầu vào vẫn cho ra đúng một dòng kết quả, cộng thêm cột do hàm cửa sổ tính.

Dấu hiệu nhận biết: luôn có mệnh đề **`OVER (...)`** đi sau tên hàm.

```
hàm(...) OVER (
    [PARTITION BY cột_chia_nhóm]
    [ORDER BY cột_sắp_xếp [ASC|DESC]]
    [khung: ROWS BETWEEN ... AND ...]
)
```

Thành phần	Vai trò	Nếu bỏ đi
`PARTITION BY`	Chia dòng thành các "phân vùng" độc lập (giống `GROUP BY` nhưng không gộp)	Cả bảng kết quả là một phân vùng
`ORDER BY`	Thứ tự dòng **bên trong** phân vùng — quyết định hạng, "phiên trước", tổng lũy kế	Thứ tự không xác định (không dùng được cho xếp hạng, `LAG`, `LEAD`)
Khung (`ROWS BETWEEN`)	Cửa sổ trượt quanh dòng hiện tại (mục 5)	Mặc định như mục 5.2

> **Ví dụ nghiệp vụ:** Phòng phân tích cần bảng "giá đóng cửa từng phiên **kèm** giá đóng cửa trung bình của chính mã đó". Không thể dùng `GROUP BY` vì sẽ mất từng phiên; hàm cửa sổ giải quyết trọn vẹn trong một câu lệnh.

1.2. So sánh với GROUP BY

Cùng một câu hỏi "giá đóng cửa trung bình và cao nhất của từng mã", hai cách cho hai kết quả khác hẳn về hình dạng.

Với GROUP BY — mỗi mã còn một dòng:

```sql
SELECT symbol,
       CAST(AVG(close_price) AS DECIMAL(18,2)) AS tb_close,
       MAX(close_price) AS cao_nhat
FROM MarketPrices
WHERE symbol IN ('FPT','VCB','HPG')
GROUP BY symbol
ORDER BY symbol;
```

```
FPT 121440.00   126000.00
HPG 27027.00    27540.00
VCB 92368.00    95680.00
```

Với hàm cửa sổ — vẫn còn từng phiên, mỗi phiên "nhìn thấy" trung bình và đỉnh của cả mã:

```sql
SELECT symbol, trade_date, close_price,
       CAST(AVG(close_price) OVER (PARTITION BY symbol) AS DECIMAL(18,2)) AS tb_close,
       MAX(close_price) OVER (PARTITION BY symbol) AS cao_nhat
FROM MarketPrices
WHERE symbol IN ('FPT','VCB','HPG') AND trade_date >= '2026-09-24'
ORDER BY symbol, trade_date;
```

```
FPT 2026-09-24  118800.00   117600.00   118800.00
FPT 2026-09-25  116400.00   117600.00   118800.00
HPG 2026-09-24  26460.00    26865.00    27270.00
HPG 2026-09-25  27270.00    26865.00    27270.00
VCB 2026-09-24  92920.00    94300.00    95680.00
VCB 2026-09-25  95680.00    94300.00    95680.00
```

Hình dưới đây đặt hai cách cạnh nhau trên cùng dữ liệu để thấy "gộp dòng" khác "giữ dòng" thế nào.

Hình minh họa: So sánh GROUP BY và Window Function (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: bên trái, `GROUP BY` nén mỗi nhóm thành một dòng nên các cột chi tiết (ngày, giá từng phiên) biến mất; bên phải, hàm cửa sổ giữ nguyên số dòng đầu vào và chỉ **thêm** một cột chứa số liệu của nhóm, lặp lại ở mọi dòng thuộc nhóm.

	`GROUP BY`	Hàm cửa sổ
Số dòng kết quả	Ít hơn (một dòng/nhóm)	Bằng số dòng đầu vào
Cột không nằm trong nhóm	Không được `SELECT` (trừ khi gộp)	`SELECT` thoải mái
Dùng khi	Báo cáo tổng hợp	Đặt số liệu nhóm cạnh chi tiết, xếp hạng, so sánh dòng với dòng

Thứ tự thực thi logic. Hàm cửa sổ được tính **sau** `WHERE`, `GROUP BY`, `HAVING`, và **trước** `ORDER BY` cuối cùng:

Bước	Mệnh đề	Ghi chú
1	`FROM` / `JOIN`	Lấy dữ liệu nguồn
2	`WHERE`	Lọc dòng — dòng bị loại **không tham gia** cửa sổ
3	`GROUP BY` / `HAVING`	Gộp nhóm (nếu có)
4	**Hàm cửa sổ** (`OVER`)	Tính trên các dòng còn lại
5	`SELECT` (biểu thức, bí danh)	Đặt tên cột
6	`ORDER BY` cuối câu, `TOP` / `LIMIT`	Sắp xếp hiển thị

> **Cảnh báo:** Hệ quả của thứ tự trên: (1) `WHERE` chạy trước nên những dòng bị lọc **không tham gia** vào cửa sổ (xem bẫy ở mục 4.2); (2) **không được đặt hàm cửa sổ trong** `WHERE` hay `GROUP BY`. Muốn lọc theo kết quả của hàm cửa sổ (ví dụ "chỉ giữ hạng `<= 3`") phải bọc câu truy vấn thành **truy vấn con ở mệnh đề FROM** (derived table) rồi lọc ở truy vấn ngoài — chúng ta sẽ dùng mẫu này rất nhiều. (Truy vấn con và CTE sẽ được học kỹ ở Buổi 5.)

1.3. Khác biệt SQL Server và MySQL

Cú pháp cơ bản **giống nhau**, các bài trong buổi này chạy nguyên văn trên cả hai. Vài điểm cần nhớ:

- Trong `OVER`, không dùng bí danh cột (alias) của `SELECT`; phải viết lại biểu thức.
- Kiểu dữ liệu kết quả có thể khác (xem mục 2.2, `AVG` trên số nguyên).
- MySQL hỗ trợ thêm mệnh đề `WINDOW w AS (...)` đặt tên cho cửa sổ để dùng lại nhiều lần; T-SQL truyền thống (trước 2022) thì không — nên viết lặp `OVER (...)`.

MySQL — đặt tên cửa sổ, viết một lần dùng nhiều lần:

```sql
SELECT trade_date, close_price,
       LAG(close_price) OVER w AS prev_close,
       SUM(volume)      OVER w AS luy_ke_kl
FROM MarketPrices
WHERE symbol = 'HPG' AND trade_date <= '2026-09-17'
WINDOW w AS (ORDER BY trade_date)
ORDER BY trade_date;
```

```
2026-09-14  27000.00    NULL    7400000
2026-09-15  26190.00    27000.00    12400000
2026-09-16  27000.00    26190.00    19800000
2026-09-17  27540.00    27000.00    25700000
```

SQL Server — viết lặp `OVER (...)` (chạy được ở mọi phiên bản):

```sql
SELECT trade_date, close_price,
       LAG(close_price) OVER (ORDER BY trade_date) AS prev_close,
       SUM(volume)      OVER (ORDER BY trade_date) AS luy_ke_kl
FROM MarketPrices
WHERE symbol = 'HPG' AND trade_date <= '2026-09-17'
ORDER BY trade_date;
```

```
2026-09-14  27000.00    NULL    7400000
2026-09-15  26190.00    27000.00    12400000
2026-09-16  27000.00    26190.00    19800000
2026-09-17  27540.00    27000.00    25700000
```

> **SQL Server vs MySQL:** Hai câu trên cho cùng kết quả. Khi chuyển từ MySQL sang SQL Server, chỉ cần thay `OVER w` bằng `OVER (ORDER BY trade_date)`.

---

2. Window Aggregate Functions: SUM / AVG / MIN / MAX / COUNT ... OVER

Các hàm tổng hợp quen thuộc thành hàm cửa sổ chỉ bằng cách thêm `OVER`.

Hàm	Ý nghĩa khi dùng với `OVER`	Ứng dụng chứng khoán
`SUM(x) OVER (...)`	Tổng của x trên cửa sổ	Tỷ trọng giá trị giao dịch, lũy kế
`AVG(x) OVER (...)`	Trung bình của x trên cửa sổ	So sánh khối lượng với mức trung bình
`MIN(x) OVER (...)` / `MAX(x) OVER (...)`	Nhỏ nhất / lớn nhất	Đỉnh/đáy của kỳ, khoảng cách tới đỉnh
`COUNT(*) OVER (...)`	Số dòng trên cửa sổ	Đếm số phiên đã có dữ liệu

2.1. `OVER ()` rỗng: cả bảng kết quả là một cửa sổ

Bài toán: trong ngày 2026-09-24, mỗi mã chiếm bao nhiêu % tổng khối lượng khớp toàn thị trường?

```sql
SELECT symbol, volume,
       SUM(volume) OVER () AS tong_ngay,
       CAST(ROUND(100.0 * volume / SUM(volume) OVER (), 2) AS DECIMAL(18,2)) AS ty_trong
FROM MarketPrices
WHERE trade_date = '2026-09-24'
ORDER BY volume DESC, symbol;
```

```
MBS 8700000 56900000    15.29
FPT 8400000 56900000    14.76
SHS 8100000 56900000    14.24
VNM 8100000 56900000    14.24
IDC 7300000 56900000    12.83
NVB 4400000 56900000    7.73
...
```

Cùng một biểu thức `SUM(volume) OVER ()` cho **cùng một con số 56.900.000 ở mọi dòng**, nên ta có thể chia trực tiếp để ra tỷ trọng — điều mà với `GROUP BY` phải viết truy vấn con hoặc tự nối bảng.

Tương tự, so sánh từng mã với **mức trung bình của cả thị trường** trong ngày:

```sql
SELECT symbol, volume,
       CASE WHEN volume * 1.0 > AVG(volume * 1.0) OVER () THEN 'Tren TB' ELSE 'Duoi TB' END
        AS so_voi_tb
FROM MarketPrices
WHERE trade_date = '2026-09-25'
ORDER BY volume DESC, symbol;
```

```
NVB 8400000 Tren TB
CEO 7800000 Tren TB
VCB 7500000 Tren TB
IDC 7000000 Tren TB
PVS 7000000 Tren TB
...
```

> **Mẹo:** Có thể đặt hàm cửa sổ **bên trong** `CASE` như trên, nhưng vẫn không thể đặt trong `WHERE`. Muốn giữ riêng các mã trên trung bình, hãy bọc thành truy vấn con rồi lọc `so_voi_tb = 'Tren TB'`.

2.2. `PARTITION BY`: cửa sổ theo nhóm

`SUM(volume) OVER (PARTITION BY symbol)` là tổng khối lượng của **riêng mã đó** trên các dòng đang xét. Ví dụ trên bảng `Trades`, đặt cạnh mỗi giao dịch tổng khối lượng khớp của mã trong giai đoạn 24–25/09:

```sql
SELECT trade_id, symbol, quantity,
       SUM(quantity) OVER (PARTITION BY symbol) AS tong_theo_ma
FROM Trades
WHERE trade_date >= '2026-09-24'
ORDER BY symbol, trade_id;
```

```
32  CEO 1000    3500
40  CEO 2500    3500
18  FPT 500 500
78  IDC 100 150
80  IDC 50  150
2   MBS 2000    2000
...
```

Bốn hàm cần nhớ: `SUM`, `AVG`, `MIN`, `MAX` (thêm `COUNT`). Ví dụ dùng `MIN` / `MAX` để đo **khoảng cách của giá hiện tại tới đỉnh 10 phiên** của chính mã đó. Vì cần cả chuỗi 10 phiên để tìm đỉnh/đáy nên phải tính trong truy vấn con rồi mới lọc ngày ở truy vấn ngoài:

```sql
SELECT symbol, close_price, dinh, day_gia,
       CAST(ROUND(100.0 * (dinh - close_price) / dinh, 2) AS DECIMAL(18,2)) AS cach_dinh_pct
FROM (
    SELECT symbol, trade_date, close_price,
           MAX(close_price) OVER (PARTITION BY symbol) AS dinh,
           MIN(close_price) OVER (PARTITION BY symbol) AS day_gia
    FROM MarketPrices
) t
WHERE trade_date = '2026-09-25'
ORDER BY cach_dinh_pct DESC, symbol;
```

```
SHS 14720.00    16320.00    14720.00    9.80
FPT 116400.00   126000.00   116400.00   7.62
CEO 20460.00    22000.00    19140.00    7.00
VNM 65960.00    70720.00    65960.00    6.73
VCS 56400.00    59400.00    54600.00    5.05
...
```

Khi có cả `PARTITION BY` lẫn `ORDER BY`, phép tính trở thành **lũy kế** (xem mục 5).

SQL Server vs MySQL — bẫy `AVG` trên số nguyên. Cột `volume` kiểu `BIGINT`. `AVG` trong T-SQL trên kiểu nguyên trả về **số nguyên (cắt bỏ phần thập phân)**, còn MySQL trả về số thập phân:

SQL Server:

```sql
SELECT symbol, volume, AVG(volume) OVER () AS tb_ngay
FROM MarketPrices WHERE trade_date = '2026-09-24' ORDER BY symbol;
```

```
CEO 2300000 4741666
FPT 8400000 4741666
HPG 200000  4741666
...
```

MySQL:

```sql
SELECT symbol, volume, AVG(volume) OVER () AS tb_ngay
FROM MarketPrices WHERE trade_date = '2026-09-24' ORDER BY symbol;
```

```
CEO 2300000 4741666.6667
FPT 8400000 4741666.6667
HPG 200000  4741666.6667
...
```

Cách viết an toàn cho cả hai: `CAST(ROUND(AVG(volume * 1.0) OVER (), 2) AS DECIMAL(18,2))` — nhân `1.0` để ép sang số thập phân, rồi `ROUND` + `CAST` để in đúng hai chữ số. Quy tắc này áp dụng cho mọi bài chấm tự động của khóa.

```sql
SELECT symbol, volume,
       CAST(ROUND(AVG(volume * 1.0) OVER (), 2) AS DECIMAL(18,2)) AS tb_ngay
FROM MarketPrices WHERE trade_date = '2026-09-24' ORDER BY symbol;
```

```
CEO 2300000 4741666.67
FPT 8400000 4741666.67
HPG 200000  4741666.67
...
```

2.3. Giá trị giao dịch: vài lưu ý nghiệp vụ

Trong buổi này, **giá trị giao dịch** của một mã trong phiên được xấp xỉ bằng `close_price * volume`. (Giá trị khớp thực tế phải cộng từng lệnh khớp; ở đây ta dùng công thức đơn giản để có dữ liệu cho cả 12 mã trong cả 10 phiên.) Tổng giá trị cả 12 mã ngày 25/09 vào cỡ 2.770 tỷ đồng, nên hãy luôn `CAST(... AS DECIMAL(18,2))` để số in ra có 2 chữ số thập phân và không bị tràn kiểu.

> **Cảnh báo:** Quên rằng `OVER ()` chỉ nhìn thấy các dòng **còn lại sau** `WHERE` — lọc ngày 25/09 xong thì `SUM(...) OVER ()` chỉ là tổng của ngày 25/09. Dùng `GROUP BY` rồi vẫn `SELECT` thêm cột chi tiết sẽ bị báo lỗi; đó là lúc nên chuyển sang hàm cửa sổ.

Bài tập 3.1c (tỷ trọng giá trị giao dịch với `SUM/AVG/MIN/MAX ... OVER ()`) dùng đúng kiến thức của mục này; các bài còn lại của nhóm 3.1 nằm ở phần xếp hạng ngay bên dưới, nên bạn sẽ làm cả nhóm sau mục 3.

---

3. Ranking Functions: ROW_NUMBER, RANK, DENSE_RANK

Ba hàm xếp hạng đều **bắt buộc có** `ORDER BY` **trong** `OVER`, khác nhau ở cách đối xử với các dòng **đồng hạng** (ties):

Hàm	Đồng hạng	Thứ hạng kế tiếp	Ví dụ với giá trị 10, 10, 8
`ROW_NUMBER()`	Vẫn được số khác nhau (tùy ý nếu không có khóa phụ)	Liền mạch	1, 2, 3
`RANK()`	Cùng hạng	**Nhảy cóc**	1, 1, 3
`DENSE_RANK()`	Cùng hạng	Liền mạch	1, 1, 2

Ngày 2026-09-23 có hai mã đồng khối lượng đứng đầu (PVS và VCS cùng 8.400.000). So sánh ba hàm:

```sql
SELECT symbol, volume,
       ROW_NUMBER() OVER (ORDER BY volume DESC, symbol) AS rn,
       RANK()       OVER (ORDER BY volume DESC)         AS rnk,
       DENSE_RANK() OVER (ORDER BY volume DESC)         AS drnk
FROM MarketPrices
WHERE trade_date = '2026-09-23'
ORDER BY rn;
```

```
PVS 8400000 1   1   1
VCS 8400000 2   1   1
NVB 8200000 3   3   2
VNM 8100000 4   4   3
MBS 7400000 5   5   4
...
```

Nhìn dòng NVB: `RANK` = 3 (vì có 2 mã hạng 1 đứng trước, hạng 2 "bị bỏ"), còn `DENSE_RANK` = 2.

3.1. Xử lý đồng hạng — chọn hàm theo nghiệp vụ

- Muốn **mỗi dòng một số duy nhất** (đánh số phiên, chọn đúng 1 dòng đầu mỗi nhóm): `ROW_NUMBER()`, **luôn thêm khóa phụ** vào `ORDER BY` (như `symbol` ở trên) để kết quả **tất định** — nếu không, hai lần chạy có thể ra hai thứ tự khác nhau.
- Muốn **"Top N" công bằng**, đồng hạng thì cùng được vào: `RANK()` (hoặc `DENSE_RANK()`), rồi lọc `hang <= N`. Với `RANK`, Top 3 có thể ra hơn 3 dòng nếu đồng hạng ở vị trí 3.
- Muốn biết mã đứng thứ *k* khác biệt về giá trị (thứ hai cao nhất, thứ ba cao nhất...): `DENSE_RANK()`.

Ví dụ `DENSE_RANK`: ngày 25/09, "mức khối lượng cao thứ tư" là bao nhiêu, mã nào đạt mức đó? Vì `DENSE_RANK` không nhảy cóc nên ba mức cao hơn (NVB, CEO, VCB) chiếm hạng 1–3, và hai mã đồng khối lượng IDC và PVS cùng đứng hạng 4 (nếu dùng `RANK` thì họ đứng hạng 4 nhưng mức "thứ năm" sẽ không tồn tại):

```sql
SELECT symbol, volume
FROM (
    SELECT symbol, volume,
           DENSE_RANK() OVER (ORDER BY volume DESC) AS d
    FROM MarketPrices
    WHERE trade_date = '2026-09-25'
) t
WHERE d = 4
ORDER BY symbol;
```

```
IDC 7000000
PVS 7000000
```

Còn khi cần **đúng một dòng mỗi nhóm** (ví dụ phiên có giá đóng cửa cao nhất của mỗi mã; đồng giá thì lấy phiên sớm hơn), dùng `ROW_NUMBER()` với khóa phụ rồi giữ `rn = 1`:

```sql
SELECT symbol, trade_date, close_price
FROM (
    SELECT symbol, trade_date, close_price,
           ROW_NUMBER() OVER (PARTITION BY symbol ORDER BY close_price DESC, trade_date) AS
        rn
    FROM MarketPrices
) t
WHERE rn = 1
ORDER BY symbol;
```

```
CEO 2026-09-14  22000.00
FPT 2026-09-17  126000.00
HPG 2026-09-17  27540.00
IDC 2026-09-24  54500.00
MBS 2026-09-25  26880.00
...
```

3.2. Xếp hạng theo từng nhóm (`PARTITION BY`) và Top N mỗi nhóm

Thêm `PARTITION BY trade_date` để xếp hạng **riêng cho từng ngày**. Vì không thể lọc hạng trong `WHERE` , ta bọc vào truy vấn con:

```sql
SELECT trade_date, symbol, volume, hang
FROM (
    SELECT trade_date, symbol, volume,
           RANK() OVER (PARTITION BY trade_date ORDER BY volume DESC) AS hang
    FROM MarketPrices
    WHERE trade_date BETWEEN '2026-09-22' AND '2026-09-24'
) t
WHERE hang <= 2
ORDER BY trade_date, hang, symbol;
```

```
2026-09-22  VCS 8600000 1
2026-09-22  NVB 8000000 2
2026-09-23  PVS 8400000 1
2026-09-23  VCS 8400000 1
2026-09-24  MBS 8700000 1
2026-09-24  FPT 8400000 2
```

Ngày 23/09 có hai mã đồng hạng 1 và ta vẫn giữ cả hai, đồng thời **không có hạng 2** — đúng tính chất của `RANK` . Đây là mẫu "**Top N theo nhóm**" mà bài 3.1 sẽ dùng.

> **Ví dụ nghiệp vụ:** Ban lãnh đạo muốn xem mỗi ngày "3 mã có giá trị giao dịch lớn nhất". Đó là Top N theo nhóm với nhóm = ngày giao dịch: `RANK() OVER (PARTITION BY trade_date ORDER BY gia_tri DESC)` rồi lọc `hang <= 3` ở truy vấn ngoài.

3.3. Phân loại theo thứ hạng: `CASE` và `NTILE`

Sau khi có hạng, ta gắn nhãn bằng `CASE WHEN` (buổi 4 sẽ học kỹ hơn):

```sql
CASE WHEN hang <= 3 THEN 'TOP3'
     WHEN hang <= 6 THEN 'KHA'
     ELSE 'YEU' END AS nhom
```

Nếu chỉ cần chia đều thành *k* nhóm, hàm `NTILE(k)` làm luôn (mã ở nhóm 1 là nhóm đầu). Với 12 mã chia 3 nhóm, mỗi nhóm 4 mã (nhớ thêm khóa phụ để tất định):

```
SELECT symbol, volume, NTILE(3) OVER (ORDER BY volume DESC, symbol) AS nhom
FROM MarketPrices WHERE trade_date = '2026-09-24'
ORDER BY volume DESC, symbol;
```

```
MBS 8700000 1
FPT 8400000 1
SHS 8100000 1
VNM 8100000 1
IDC 7300000 2
NVB 4400000 2
...
```

Lưu ý `NTILE` **cắt cứng theo số dòng**: SHS và VNM đồng khối lượng nhưng nếu nằm ở ranh giới nhóm vẫn có thể bị tách ra hai nhóm. Khi cần công bằng với đồng hạng, hãy dùng `RANK` + `CASE`.

Sai lầm thường gặp: dùng `ROW_NUMBER()` để "Top N" khi đề nói "đồng hạng cùng vào" (sẽ bỏ sót); quên `ORDER BY` trong `OVER` của hàm xếp hạng (lỗi cú pháp); quên khóa phụ khiến kết quả không ổn định giữa các lần chạy.

Thực hành ngay: bài 3.1

Vừa học xong tổng hợp `OVER` và ba hàm xếp hạng, hãy làm nhóm **bài 3.1** để đánh số phiên, so sánh `ROW_NUMBER/RANK/DENSE_RANK`, tính tỷ trọng, lấy Top 3 mỗi ngày và phân loại cổ phiếu theo hạng.

- **Bài 3.1** — Đánh số phiên giao dịch theo từng mã *(độ khó: dễ)*
- **Bài 3.1** — Xếp hạng khối lượng: ROW_NUMBER, RANK, DENSE_RANK *(độ khó: dễ)*
- **Bài 3.1** — Tỷ trọng giá trị giao dịch với hàm tổng hợp OVER *(độ khó: dễ)*
- **Bài 3.1** — Top 3 mã theo giá trị giao dịch từng ngày *(độ khó: trung bình)*
- **Bài 3.1** — Phân loại cổ phiếu theo thứ hạng khối lượng *(độ khó: trung bình)*

---

4. LAG và LEAD — phân tích biến động

`LAG` lấy giá trị của **dòng đứng trước**, `LEAD` lấy giá trị của **dòng đứng sau** trong cùng phân vùng, theo thứ tự `ORDER BY` của cửa sổ.

```
LAG (biểu_thức [, n [, giá_trị_mặc_định]]) OVER (PARTITION BY ... ORDER BY ...)
LEAD(biểu_thức [, n [, giá_trị_mặc_định]]) OVER (PARTITION BY ... ORDER BY ...)
```

Tham số	Ý nghĩa	Mặc định
`biểu_thức`	Cột/biểu thức cần lấy từ dòng trước/sau	(bắt buộc)
`n`	Lùi/tiến bao nhiêu dòng	1
`giá_trị_mặc_định`	Giá trị dùng khi không có dòng trước/sau	`NULL`

Ví dụ với VNM, 5 phiên cuối:

```
SELECT trade_date, close_price,
       LAG(close_price)  OVER (ORDER BY trade_date) AS prev_close,
       LEAD(close_price) OVER (ORDER BY trade_date) AS next_close
FROM MarketPrices
WHERE symbol = 'VNM' AND trade_date >= '2026-09-21'
ORDER BY trade_date;
```

```
2026-09-21  66640.00    NULL    68000.00
2026-09-22  68000.00    66640.00    66640.00
2026-09-23  66640.00    68000.00    67320.00
2026-09-24  67320.00    66640.00    65960.00
2026-09-25  65960.00    67320.00    NULL
```

Phiên đầu không có phiên trước (`NULL`), phiên cuối không có phiên sau (`NULL`). Nếu dùng tham số thứ ba, `LAG(close_price, 2, 0)` lùi 2 phiên và thay `NULL` bằng `0` :

```
SELECT trade_date, close_price,
       LAG(close_price, 2, 0) OVER (ORDER BY trade_date) AS gia_2_phien_truoc
FROM MarketPrices
WHERE symbol = 'VNM' AND trade_date >= '2026-09-21'
ORDER BY trade_date;
```

```
2026-09-21  66640.00    0.00
2026-09-22  68000.00    0.00
2026-09-23  66640.00    66640.00
2026-09-24  67320.00    68000.00
2026-09-25  65960.00    66640.00
```

4.1. Mức tăng/giảm và % biến động so với phiên trước

Công thức chuẩn trong ngành:

- Mức thay đổi = `close − prev_close`
- % thay đổi = `100 * (close − prev_close) / prev_close`

4.2. Bẫy lớn: lọc ngày trước khi gọi LAG

Nhiều bạn muốn xem % thay đổi của **riêng phiên 25/09** và viết ngay `WHERE trade_date = '2026-09-25'`. Vì `WHERE` chạy **trước** hàm cửa sổ, chỉ còn đúng một dòng trong cửa sổ và `LAG` trả về `NULL`. Ví dụ dưới đây lọc từ 24/09 trở đi nên phiên 24/09 mất phiên trước:

```
-- SAI: prev_close của phiên đầu luôn NULL vì phiên 23/09 đã bị WHERE loại
SELECT trade_date, close_price,
       LAG(close_price) OVER (ORDER BY trade_date) AS prev_close
FROM MarketPrices
WHERE symbol = 'VNM' AND trade_date >= '2026-09-24'
ORDER BY trade_date;
```

```
2026-09-24  67320.00    NULL
2026-09-25  65960.00    67320.00
```

Cách đúng: tính `LAG` trên **cả chuỗi thời gian** trong truy vấn con, rồi lọc ngày ở truy vấn ngoài:

```
SELECT trade_date, close_price, prev_close,
       CAST(ROUND(100.0 * (close_price - prev_close) / prev_close, 2) AS DECIMAL(18,2)) AS
         pct
FROM (
    SELECT trade_date, close_price,
           LAG(close_price) OVER (ORDER BY trade_date) AS prev_close
    FROM MarketPrices
    WHERE symbol = 'VNM'
) t
WHERE trade_date >= '2026-09-23'
ORDER BY trade_date;
```

```
2026-09-23  66640.00    68000.00    -2.00
2026-09-24  67320.00    66640.00    1.02
2026-09-25  65960.00    67320.00    -2.02
```

> **Cảnh báo:** `WHERE` lọc ngày **trước** `LAG` = mất phiên trước. Luôn tính hàm cửa sổ trên tập dữ liệu đầy đủ ở truy vấn con, rồi lọc ở truy vấn ngoài.

4.3. Phân vùng theo mã khi có nhiều mã

Với nhiều mã, **bắt buộc** `PARTITION BY symbol` — nếu không, phiên đầu của mã B sẽ "lấy giá phiên trước" là phiên cuối của mã A, một lỗi logic rất khó thấy. Cũng nhớ rằng "phiên trước" là **dòng đứng trước theo thứ tự**, không phải ngày liền kề: nếu mã nghỉ giao dịch, phiên trước là phiên có dữ liệu gần nhất.

Ứng dụng thường gặp: gắn nhãn phiên tăng/giảm/đi ngang bằng `CASE` trên kết quả `LAG` (phiên đầu không có phiên trước nên nhãn là `NULL`):

```sql
SELECT trade_date, close_price, prev_close,
       CASE WHEN close_price > prev_close THEN 'TANG'
            WHEN close_price < prev_close THEN 'GIAM'
            WHEN close_price = prev_close THEN 'DUNG' END AS xu_huong
FROM (
    SELECT trade_date, close_price,
           LAG(close_price) OVER (PARTITION BY symbol ORDER BY trade_date) AS prev_close
    FROM MarketPrices
    WHERE symbol = 'FPT'
) t
WHERE trade_date <= '2026-09-17'
ORDER BY trade_date;
```

```
2026-09-14  123600.00   NULL    NULL
2026-09-15  120000.00   123600.00   GIAM
2026-09-16  122400.00   120000.00   TANG
2026-09-17  126000.00   122400.00   TANG
```

Sai lầm thường gặp: chia cho `prev_close = 0` hoặc `NULL` (kết quả `NULL` /lỗi chia 0 — hãy lọc `prev_close IS NOT NULL`), so sánh `close_price > prev_close` với `NULL` (luôn không đúng), quên `PARTITION BY`.

> **Mẹo:** Một số hàm "họ hàng" cùng cú pháp: `FIRST_VALUE`, `LAST_VALUE` (giá trị dòng đầu/cuối của khung) — sẽ học ở mục 5.5 vì chúng phụ thuộc vào khung cửa sổ.

Thực hành ngay: bài 3.2

Vừa học xong `LAG` / `LEAD` và bẫy "lọc ngày trước khi tính", hãy làm nhóm **bài 3.2** để tính giá phiên trước/sau của HPG, % biến động, đếm phiên tăng/giảm và tìm phiên giảm sâu nhất của từng mã.

- **Bài 3.2** — Giá đóng cửa phiên trước và phiên sau của HPG *(độ khó: dễ)*
- **Bài 3.2** — % biến động giá so với phiên trước *(độ khó: trung bình)*
- **Bài 3.2** — Đếm phiên tăng, giảm, đi ngang của từng mã *(độ khó: trung bình)*
- **Bài 3.2** — Phiên biến động xấu nhất của từng mã *(độ khó: khó)*

---

5. Window Frame và Moving Average

5.1. Khung cửa sổ

Khi có `ORDER BY`, ta còn kiểm soát được **những dòng nào** của phân vùng được đưa vào phép tính cho dòng hiện tại bằng **khung**:

```sql
ROWS BETWEEN <đầu> AND <cuối>
```

Với `<đầu>` / `<cuối>` là một trong:

Từ khóa	Nghĩa
`UNBOUNDED PRECEDING`	Từ dòng đầu tiên của phân vùng
`n PRECEDING`	`n` dòng trước dòng hiện tại
`CURRENT ROW`	Dòng hiện tại
`n FOLLOWING`	`n` dòng sau dòng hiện tại
`UNBOUNDED FOLLOWING`	Tới dòng cuối của phân vùng

Hình dưới minh họa khung `ROWS BETWEEN 2 PRECEDING AND CURRENT ROW` — chính là khung của đường trung bình động MA3.

Hình minh họa: Khung cửa sổ (Window Frame) (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: với mỗi dòng hiện tại (được tô đậm), khung "trượt" bao gồm nó và **2 dòng đứng ngay trước**; khi dòng hiện tại dịch xuống một phiên, khung cũng dịch theo. Ở hai dòng đầu, khung bị "cắt" ở biên phân vùng nên chỉ có 1–2 dòng.

Ví dụ nhanh trên mã PVI, các phiên đến 18/09 — cột `luy_ke` (từ đầu đến nay) và `tong_3_phien` (3 phiên gần nhất):

```
SELECT trade_date, volume,
       SUM(volume) OVER (ORDER BY trade_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT
         ROW) AS luy_ke,
       SUM(volume) OVER (ORDER BY trade_date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
         AS tong_3_phien
FROM MarketPrices
WHERE symbol = 'PVI' AND trade_date <= '2026-09-18'
ORDER BY trade_date;
```

```
2026-09-14  1100000 1100000 1100000
2026-09-15  6500000 7600000 7600000
2026-09-16  800000  8400000 8400000
2026-09-17  500000  8900000 7800000
2026-09-18  6000000 14900000    7300000
```

Chú ý dòng đầu: chưa đủ 3 phiên nên `tong_3_phien` chỉ cộng những phiên **hiện có** (khung bị "cắt" ở biên phân vùng).

5.2. Mặc định của khung — và vì sao nên viết rõ `ROWS`

Nếu có `ORDER BY` mà **không** viết khung, chuẩn SQL dùng `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`. Với `RANGE`, các dòng **có cùng giá trị sắp xếp (đồng hạng) được coi là cùng một "điểm"** và cùng được cộng vào. Trên dữ liệu ngày 25/09 có đồng khối lượng:

```sql
SELECT symbol, volume,
       SUM(volume) OVER (ORDER BY volume DESC) AS mac_dinh
FROM MarketPrices
WHERE trade_date = '2026-09-25' AND symbol IN ('NVB','IDC','PVS','VNM','MBS','SHS')
ORDER BY volume DESC, symbol;
```

```
NVB 8400000 8400000
IDC 7000000 22400000
PVS 7000000 22400000
VNM 5400000 27800000
MBS 5000000 37800000
SHS 5000000 37800000
```

Ở kết quả trên, IDC và PVS cùng khối lượng nên **cùng nhận tổng 22.400.000**; MBS và SHS cũng vậy. Còn `ROWS` đếm **từng dòng vật lý** nên cho tổng lũy kế "đúng nghĩa từng dòng" — nhưng thứ tự giữa hai dòng đồng giá trị phải được **chốt bằng khóa phụ**, nếu không SQL Server và MySQL có thể sắp IDC/PVS theo thứ tự khác nhau và cho ra hai kết quả lũy kế khác nhau ở hai dòng này. Khuyến nghị cho khóa học:

> **Lưu ý:** Luôn viết tường minh `ROWS BETWEEN ... AND ...` và đặt `ORDER BY` **có khóa duy nhất** (ví dụ `ORDER BY trade_date` khi đã `PARTITION BY symbol` — vì `(symbol, trade_date)` là khóa chính).

Trên SQL Server, `ROWS` còn nhanh hơn `RANGE` đáng kể với bảng lớn. Hai hệ đều hỗ trợ `ROWS` và `RANGE` trong buổi này.

5.3. Giá trị lũy kế

Tổng lũy kế = tổng từ đầu đến dòng hiện tại: `SUM(x) OVER (PARTITION BY k ORDER BY t ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)`. Ứng dụng: dòng tiền vào/ra tích lũy, khối lượng giao dịch tích lũy trong kỳ, giá trị danh mục theo thời gian.

Có thể kết hợp `GROUP BY` **rồi** hàm cửa sổ trên kết quả đã gộp: hàm cửa sổ chạy sau `GROUP BY`, nên được phép viết hàm tổng hợp *bên trong* hàm cửa sổ. Ví dụ giá trị giao dịch từng ngày trên bảng `Trades` và lũy kế toàn công ty:

```sql
SELECT trade_date,
       SUM(quantity * price) AS gia_tri_ngay,
       SUM(SUM(quantity * price)) OVER (ORDER BY trade_date ROWS BETWEEN UNBOUNDED PRECEDING
         AND CURRENT ROW) AS luy_ke
FROM Trades
WHERE trade_date >= '2026-09-14'
GROUP BY trade_date
ORDER BY trade_date;
```

```
2026-09-14  87800000.00 87800000.00
2026-09-15  473000000.00    560800000.00
2026-09-16  377250000.00    938050000.00
2026-09-17  189000000.00    1127050000.00
...
```

> **Ví dụ nghiệp vụ:** Bộ phận thanh toán theo dõi "giá trị giao dịch lũy kế trong tuần" của từng mã để cảnh báo mã tập trung dòng tiền: `SUM(gia_tri) OVER (PARTITION BY symbol ORDER BY trade_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)`.

5.4. Đường trung bình động (Moving Average)

Trung bình động *N* phiên (MA*N*) làm mượt dao động ngắn hạn để nhìn xu hướng: trung bình giá của phiên hiện tại và *N−1* phiên liền trước.

```sql
AVG(close_price) OVER (ORDER BY trade_date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)  -- MA3
AVG(close_price) OVER (ORDER BY trade_date ROWS BETWEEN 4 PRECEDING AND CURRENT ROW)  -- MA5
```

Đường	Khung	Số dòng trong khung	Nhạy với biến động
MA3	`2 PRECEDING AND CURRENT ROW`	3	Cao (bám sát giá)
MA5	`4 PRECEDING AND CURRENT ROW`	5	Thấp hơn (mượt hơn)

Ví dụ với VCB: cột `so_phien` cho biết khung đang có bao nhiêu dòng, để biết MA ở đầu chuỗi chưa "đủ" N phiên:

```sql
SELECT trade_date, close_price,
       COUNT(*) OVER (ORDER BY trade_date ROWS BETWEEN 4 PRECEDING AND CURRENT ROW) AS
         so_phien,
       CAST(AVG(close_price) OVER (ORDER BY trade_date ROWS BETWEEN 2 PRECEDING AND CURRENT
         ROW) AS DECIMAL(18,2)) AS ma3,
       CAST(AVG(close_price) OVER (ORDER BY trade_date ROWS BETWEEN 4 PRECEDING AND CURRENT
         ROW) AS DECIMAL(18,2)) AS ma5
FROM MarketPrices
WHERE symbol = 'VCB' AND trade_date <= '2026-09-18'
ORDER BY trade_date;
```

```
2026-09-14  93840.00    1   93840.00    93840.00
2026-09-15  91080.00    2   92460.00    92460.00
2026-09-16  92000.00    3   92306.67    92306.67
2026-09-17  90160.00    4   91080.00    91770.00
2026-09-18  92920.00    5   91693.33    92000.00
```

Khung cũng có thể **có tâm**, dùng cả phiên sau (`1 PRECEDING` đến `1 FOLLOWING`) — hữu ích để làm mượt dữ liệu lịch sử khi phân tích, nhưng không dùng được để "dự báo" vì nhìn trước tương lai:

```sql
SELECT trade_date, close_price,
       CAST(AVG(close_price) OVER (ORDER BY trade_date ROWS BETWEEN 1 PRECEDING AND 1
        FOLLOWING) AS DECIMAL(18,2)) AS tb_co_tam
FROM MarketPrices
WHERE symbol = 'VCB' AND trade_date <= '2026-09-18'
ORDER BY trade_date;
```

```
2026-09-14  93840.00    92460.00
2026-09-15  91080.00    92306.67
2026-09-16  92000.00    91080.00
2026-09-17  90160.00    91693.33
2026-09-18  92920.00    91540.00
```

Đọc tín hiệu: nếu MA ngắn hạn (MA3) cắt lên trên MA dài hạn (MA5) thường được xem là tín hiệu **tăng**, cắt xuống là tín hiệu **giảm** — đây chính là ý tưởng của bài 3.3.

Sai lầm thường gặp:

- Quên `PARTITION BY symbol` khi tính MA cho nhiều mã: MA của mã này "dính" giá mã khác.
- Nhầm `ROWS BETWEEN 3 PRECEDING` với MA3: `3 PRECEDING AND CURRENT ROW` gồm **4** dòng; MA3 phải là `2 PRECEDING`.
- Tin rằng MA ở các phiên đầu chưa đủ *N* phiên là "đúng" — thực tế đó là trung bình của các phiên hiện có; nếu cần chỉ tính khi đủ dữ liệu, thêm `COUNT(*) OVER (...)` hoặc `ROW_NUMBER()` rồi lọc.

5.5. `FIRST_VALUE` và `LAST_VALUE` — bẫy của khung mặc định

`FIRST_VALUE(x)` trả giá trị của dòng **đầu khung**, `LAST_VALUE(x)` trả giá trị của dòng **cuối khung**. Bẫy kinh điển: với khung mặc định (kết thúc ở `CURRENT ROW`), `LAST_VALUE` chỉ trả về **chính dòng hiện tại**. Muốn lấy giá cuối kỳ phải mở rộng khung tới `UNBOUNDED FOLLOWING`:

```sql
SELECT trade_date, close_price,
       FIRST_VALUE(close_price) OVER (ORDER BY trade_date) AS gia_dau_ky,
       LAST_VALUE(close_price)  OVER (ORDER BY trade_date) AS last_mac_dinh,
       LAST_VALUE(close_price)  OVER (ORDER BY trade_date
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS gia_cuoi_ky
FROM MarketPrices
WHERE symbol = 'FPT' AND trade_date <= '2026-09-17'
ORDER BY trade_date;
```

```
2026-09-14  123600.00   123600.00   123600.00   126000.00
2026-09-15  120000.00   123600.00   120000.00   126000.00
2026-09-16  122400.00   123600.00   122400.00   126000.00
2026-09-17  126000.00   123600.00   126000.00   126000.00
```

Cột `last_mac_dinh` chỉ lặp lại `close_price` của từng dòng — kết quả "sai" so với mong đợi; cột `gia_cuoi_ky` mới đúng là giá của phiên cuối trong tập đang xét.

> **Mẹo:** Nhờ `FIRST_VALUE` ta tính được "hiệu suất kể từ đầu kỳ" = `100.0 * (close_price - gia_dau_ky) / gia_dau_ky` mà không cần tự nối bảng.

Thực hành ngay: bài 3.3

Vừa học xong khung cửa sổ, lũy kế và trung bình động, hãy làm nhóm **bài 3.3** để tính giá trị giao dịch lũy kế của từng mã, đường MA3/MA5 của FPT và nhận diện tín hiệu xu hướng.

- **Bài 3.3** — Giá trị giao dịch lũy kế của từng mã *(độ khó: dễ)*
- **Bài 3.3** — Đường trung bình động MA3 và MA5 của FPT *(độ khó: trung bình)*
- **Bài 3.3** — Tín hiệu xu hướng từ MA3 và MA5 *(độ khó: khó)*

---

6. Tổng kết và Q&A

Tóm tắt

Chủ đề	Cú pháp chính	Ghi nhớ
Hàm cửa sổ	`f(...) OVER (PARTITION BY ... ORDER BY ...)`	Không gộp dòng, chạy sau WHERE/GROUP BY
Tổng hợp	`SUM/AVG/MIN/MAX/COUNT ... OVER`	`OVER ()` = cả tập kết quả; `AVG` số nguyên khác nhau giữa hai hệ
Xếp hạng	`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `NTILE`	Đồng hạng: RANK nhảy cóc, DENSE_RANK liền mạch; thêm khóa phụ
Top N theo nhóm	Truy vấn con + `RANK() OVER (PARTITION BY ...)` + `WHERE hang <= N`	Không lọc hạng trực tiếp trong WHERE
Dịch dòng	`LAG`, `LEAD`	Cần `ORDER BY`; đầu/cuối là NULL; tính trước khi lọc ngày
Khung	`ROWS BETWEEN ... AND ...`	Viết rõ `ROWS`; MA N phiên = `N-1 PRECEDING`
Đầu/cuối khung	`FIRST_VALUE`, `LAST_VALUE`	`LAST_VALUE` cần khung tới `UNBOUNDED FOLLOWING`

Q&A thường gặp

- **Có thể dùng hai hàm cửa sổ với hai `OVER` khác nhau trong cùng một `SELECT` không?** Có, như MA3 và MA5 ở trên.
- **Hàm cửa sổ có chậm không?** Thường nhanh hơn tự nối bảng. Trên bảng lớn nên có chỉ mục theo `(cột PARTITION BY, cột ORDER BY)`.
- **Có dùng `ORDER BY` cuối câu lệnh khác với `ORDER BY` trong `OVER` không?** Có: cái trong `OVER` chỉ định thứ tự để **tính**, cái cuối câu quyết định thứ tự **hiển thị**; hai cái độc lập.
- **Khi nào dùng `GROUP BY`, khi nào dùng hàm cửa sổ?** Muốn báo cáo tổng hợp (mỗi nhóm một dòng) dùng `GROUP BY`; muốn giữ chi tiết cạnh số liệu nhóm, xếp hạng, so sánh phiên trước/sau, lũy kế: hàm cửa sổ.

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Lọc ngày trong `WHERE` rồi mới gọi `LAG` / `SUM OVER`	Mất phiên trước, tổng chỉ tính trên phần đã lọc	Tính hàm cửa sổ ở truy vấn con, lọc ở truy vấn ngoài
Đặt hàm cửa sổ trong `WHERE`	Lỗi cú pháp	Bọc thành truy vấn con rồi lọc cột hạng
Quên `PARTITION BY symbol` khi có nhiều mã	Giá/khung "dính" sang mã khác	Luôn chia phân vùng theo mã
`ROW_NUMBER` cho Top N có đồng hạng	Bỏ sót mã đồng hạng	Dùng `RANK()` hoặc `DENSE_RANK()`
`ORDER BY` không có khóa duy nhất	Kết quả khác nhau giữa lần chạy/giữa hai hệ	Thêm khóa phụ (`symbol` , `trade_date`)
`3 PRECEDING` cho MA3	Trung bình 4 phiên	MA3 = `2 PRECEDING AND CURRENT ROW`
`LAST_VALUE` với khung mặc định	Trả lại chính dòng hiện tại	Khung `... AND UNBOUNDED FOLLOWING`
`AVG` số nguyên không ép kiểu	SQL Server cắt phần thập phân	Nhân `1.0` rồi `CAST(ROUND(...) AS DECIMAL(18,2))`

Bài tập thực hành của buổi

Các bài đều chấm tự động trên cả SQL Server và MySQL và dùng ngày cố định 2026-09-25 làm "hôm nay". Nếu bạn chưa làm hết các nhóm bài đã gắn trong bài giảng, đây là danh sách đầy đủ của buổi.

- **Bài 3.1** — Đánh số phiên giao dịch theo từng mã *(độ khó: dễ)*
- **Bài 3.1** — Xếp hạng khối lượng: ROW_NUMBER, RANK, DENSE_RANK *(độ khó: dễ)*
- **Bài 3.1** — Tỷ trọng giá trị giao dịch với hàm tổng hợp OVER *(độ khó: dễ)*
- **Bài 3.1** — Top 3 mã theo giá trị giao dịch từng ngày *(độ khó: trung bình)*
- **Bài 3.1** — Phân loại cổ phiếu theo thứ hạng khối lượng *(độ khó: trung bình)*
- **Bài 3.2** — Giá đóng cửa phiên trước và phiên sau của HPG *(độ khó: dễ)*
- **Bài 3.2** — % biến động giá so với phiên trước *(độ khó: trung bình)*
- **Bài 3.2** — Đếm phiên tăng, giảm, đi ngang của từng mã *(độ khó: trung bình)*
- **Bài 3.2** — Phiên biến động xấu nhất của từng mã *(độ khó: khó)*
- **Bài 3.3** — Giá trị giao dịch lũy kế của từng mã *(độ khó: dễ)*
- **Bài 3.3** — Đường trung bình động MA3 và MA5 của FPT *(độ khó: trung bình)*
- **Bài 3.3** — Tín hiệu xu hướng từ MA3 và MA5 *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Gợi ý mở rộng

- Thử `PERCENT_RANK()` và `CUME_DIST()` để xếp hạng theo phần trăm; thử `LAST_VALUE` với khung `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` cho nhiều mã cùng lúc.
- Tự tính **độ biến động (volatility)** của từng mã: độ lệch chuẩn của % thay đổi 5 phiên gần nhất (`STDEV` trong T-SQL, `STDDEV_SAMP` trong MySQL — hai tên hàm khác nhau).

- Tính chỉ báo giao cắt vàng/giao cắt tử thần: tìm phiên mà MA3 vượt MA5 lần đầu (kết hợp `LAG` trên chính MA3 - MA5).
- Chuẩn bị cho Buổi 4: các bài này dùng `CASE`, `CAST`, `ROUND` — buổi tới sẽ học hệ thống các hàm xử lý chuỗi, ngày giờ, toán học và điều kiện.

Thực hành trên Windows: so sánh GROUP BY với Window Function

1. Kiểm tra phiên bản. Window Functions cần SQL Server 2012 trở lên và MySQL 8.0 trở lên:

```
SELECT @@VERSION;        -- SQL Server
-- SELECT VERSION();     -- MySQL
```

Bản Express mới cài ở Buổi 1 đã đủ điều kiện.

2. Mở hai cửa sổ cạnh nhau để so sánh. Soạn câu `GROUP BY` ở cửa sổ 1 và câu `OVER` ở cửa sổ 2. Chuột phải vào tab của cửa sổ 2 → **New Vertical Tab Group** để hai kết quả nằm cạnh nhau.

```
-- Cửa sổ 1: gộp dòng
SELECT symbol, AVG(close_price) AS tb_close
FROM MarketPrices
GROUP BY symbol;

-- Cửa sổ 2: giữ nguyên từng dòng, thêm cột trung bình
SELECT symbol, trade_date, close_price,
       AVG(close_price) OVER (PARTITION BY symbol) AS tb_close
FROM MarketPrices;
```

Quan sát: cửa sổ 1 mỗi mã **một dòng**, cửa sổ 2 giữ **mọi phiên** và lặp lại giá trị trung bình.

3. Đọc lỗi cho nhanh. Bật số dòng: **Tools → Options → Text Editor → All Languages → General → Line numbers**. Khi SSMS báo lỗi *Line 5: Incorrect syntax near...*, bạn nhảy ngay đến dòng 5 (hoặc **Ctrl+G**).

4. Bài thử có chủ đích. Cố tình đặt `ORDER BY` trong `OVER` thiếu `PARTITION BY` rồi so sánh với có `PARTITION BY`, ghi lại số dòng và giá trị `ROW_NUMBER()` bắt đầu lại từ 1 ở mỗi mã như thế nào.

```
SELECT symbol, trade_date, close_price,
       ROW_NUMBER() OVER (PARTITION BY symbol ORDER BY trade_date) AS stt
FROM MarketPrices
WHERE symbol IN ('FPT','HPG')
ORDER BY symbol, trade_date;
```

5. Chú ý về thứ tự. Kết quả của `OVER` không tự được sắp xếp theo `ORDER BY` trong `OVER`; muốn lưới hiển thị theo thứ tự thì phải có `ORDER BY` ở cuối câu lệnh.

Với MySQL Workbench: dùng **Ctrl+T** để mở tab mới; mỗi tab chạy trong một kết nối riêng. Window Functions chỉ chạy được trên MySQL 8.

Tự kiểm tra: làm lại **bài 3.2** (% biến động giá so với phiên trước) trên SSMS, so với kết quả mong đợi, rồi nộp trên Codelab.

Buổi 4 — Kiểu dữ liệu và hàm xử lý dữ liệu

Về mục lục

Dữ liệu trong công ty chứng khoán hiếm khi "sạch" ngay để đưa vào báo cáo: mã khách hàng cần tách phần số, tên cần chuẩn hóa chữ hoa, ngày đăng ký cần đổi thành "đã tham gia bao nhiêu ngày", số dư cần phân loại VIP/Thường, giá cần làm tròn... Buổi này trang bị **bộ hàm dựng sẵn** để biến dữ liệu thô thành thông tin dùng được — và quan trọng không kém, **biết chỗ SQL Server và MySQL khác nhau** để không bị "ngã" khi chuyển hệ.

Toàn bộ ví dụ chạy trên bộ dữ liệu mẫu của khóa (xem mục **"Bộ dữ liệu mẫu của khóa học"** ở Buổi 1). "Hôm nay" cố định là **2026-09-25**: trong tài liệu và bài chấm, ta **viết thẳng** `'2026-09-25'` thay vì `GETDATE()` / `NOW()`, để kết quả lặp lại được ở mọi lần chạy.

> **Mẹo:** Buổi này thiên về **bảng tra cứu**. Hãy giữ bảng so sánh hai hệ ở từng mục bên cạnh khi làm bài, và bấm "Chạy thử" trên từng ví dụ để thấy kết quả thật.

Mục tiêu buổi học

- Sau buổi này bạn có thể chọn đúng kiểu dữ liệu (số nguyên, số thực, chuỗi, ngày giờ, boolean) cho dữ liệu chứng khoán và chuyển kiểu bằng `CAST`.
- Sau buổi này bạn có thể dùng hàm chuỗi: độ dài, cắt/nối chuỗi, chữ hoa/thường, `TRIM`, `REPLACE`, tìm vị trí ký tự.
- Sau buổi này bạn có thể dùng hàm ngày giờ: cộng/trừ ngày, chênh lệch ngày, `YEAR/MONTH/DAY`, định dạng ngày; tính số ngày tham gia, tuổi.
- Sau buổi này bạn có thể dùng hàm toán học: `ABS`, `ROUND`, `CEILING`, `FLOOR`, `POWER`, `SQRT`, `MOD`.
- Sau buổi này bạn có thể dùng hàm điều kiện `CASE`, `IF/IIF`, `COALESCE/IFNULL/ISNULL` và nhận biết khác biệt giữa SQL Server và MySQL.

Bản đồ buổi học

Buổi học 3 giờ (180 phút), khoảng 32% lý thuyết + demo (~57 phút) và 68% thực hành (~123 phút, gồm cả thời gian chữa bài). Mỗi phần lý thuyết được **xen kẽ ngay** bằng bài thực hành tương ứng: học đến đâu làm đến đó. Phần lý thuyết chủ yếu là bảng tra cứu nên được giảng gọn — hãy dùng bảng so sánh hai hệ làm tài liệu mở sẵn khi làm bài.

Phần	Nội dung	Thời lượng gợi ý
1	Các kiểu dữ liệu phổ biến, `CAST`	14 phút (lý thuyết + demo)
2	Hàm xử lý chuỗi	10 phút (lý thuyết + demo)
Thực hành	**Bài 4.1** — Xử lý chuỗi (2 bài)	21 phút
3	Hàm xử lý ngày giờ	11 phút (lý thuyết + demo)
Thực hành	**Bài 4.2** — Xử lý ngày giờ (5 bài)	40 phút
4	Hàm toán học	6 phút (lý thuyết + demo)
Thực hành	**Bài 4.4** — Hàm toán học (bổ sung) (2 bài)	24 phút
5	Hàm điều kiện: `CASE`, `IF/IIF`, `COALESCE`	9 phút (lý thuyết + demo)
Thực hành	**Bài 4.3** — Sử dụng IF và CASE (3 bài, có 2 bài khó)	38 phút
6	Tổng kết và bảng tra cứu hai hệ	7 phút
	Tổng	**180 phút** — lý thuyết + demo ~57 phút, thực hành ~123 phút (68%)

---

1. Các kiểu dữ liệu phổ biến

Chọn kiểu dữ liệu đúng quyết định độ chính xác của phép tính, dung lượng lưu trữ và cả hiệu năng truy vấn.

1.1. Số nguyên

Kiểu	SQL Server	MySQL	Dùng cho
`TINYINT`	0..255	-128..127 (hoặc `UNSIGNED` 0..255)	Cờ trạng thái nhỏ, mã loại
`SMALLINT`	±32.767	±32.767	Số lượng nhỏ
`INT`	±2,1 tỷ	±2,1 tỷ	Khóa chính, số lượng lệnh (`quantity`)
`BIGINT`	±9,2 × 10^18	±9,2 × 10^18	Khối lượng giao dịch tích lũy (`volume`), số dư cực lớn

Trong bộ dữ liệu: `Customers.customer_id`, `Orders.quantity` là `INT`; `MarketPrices.volume` là `BIGINT` vì tổng khối lượng cả thị trường có thể vượt 2,1 tỷ.

Phép chia số nguyên là bẫy kinh điển. Cùng một biểu thức `7 / 2`, hai hệ cho hai kết quả:

SQL Server — cả hai vế là số nguyên thì kết quả nguyên, cắt phần thập phân:

```sql
SELECT 7 / 2 AS ket_qua;
```

```
3
```

MySQL — toán tử `/` luôn cho số thập phân (muốn chia nguyên dùng `DIV`):

```sql
SELECT 7 / 2 AS ket_qua;
```

```
3.5000
```

Để có cùng kết quả ở cả hai hệ, viết một vế dạng thập phân (`7 / 2.0` , hoặc `100.0 * a / b`) rồi ép kiểu kết quả:

```sql
SELECT CAST(7 / 2.0 AS DECIMAL(18,2)) AS chia_thap_phan,
       CAST(100.0 * 3 / 8 AS DECIMAL(18,2)) AS phan_tram;
```

```
3.50    37.50
```

> **Lưu ý: Quy tắc của khóa học:** mọi tỷ lệ/phần trăm đều nhân `100.0` (hoặc chia `x.0`) và bọc `CAST(ROUND(..., 2) AS DECIMAL(18,2))` .

1.2. Số thực

Kiểu	Bản chất	Dùng khi
`DECIMAL(p, s)` / `NUMERIC(p, s)`	**Số thập phân chính xác**: tối đa `p` chữ số, trong đó `s` chữ số sau dấu phẩy	**Tiền, giá, số dư, tỷ lệ** — luôn ưu tiên
`FLOAT` , `DOUBLE` (MySQL) / `FLOAT` , `REAL` (SQL Server)	**Số dấu phẩy động xấp xỉ** (sai số ~1e-15)	Tính toán khoa học, thống kê (căn, lũy thừa)

Ví dụ của khóa: `Accounts.balance` là `DECIMAL(18,2)` , `MarketPrices.close_price` là `DECIMAL(12,2)` , `CustomerStats.total_trade_value` là `DECIMAL(20,2)` .

Không bao giờ lưu tiền bằng `FLOAT` . Hãy so sánh `0.1 + 0.2` với `0.3` bằng hai kiểu (hằng có hậu tố `e0` là số dấu phẩy động):

```sql
SELECT CASE WHEN 0.1e0 + 0.2e0 = 0.3e0 THEN 'bang nhau' ELSE 'khac nhau' END AS kieu_float,
       CASE WHEN CAST(0.1 AS DECIMAL(10,2)) + CAST(0.2 AS DECIMAL(10,2)) = CAST(0.3 AS
        DECIMAL(10,2))
            THEN 'bang nhau' ELSE 'khac nhau' END AS kieu_decimal;
```

```
khac nhau   bang nhau
```

Dấu phẩy động cho kết quả **khác nhau** — sai lệch nhỏ nhưng cộng hàng triệu dòng sẽ thành sai lệch tiền thật. Chỉ dùng `FLOAT/DOUBLE` cho bước tính trung gian như `SQRT`, `POWER` (mục 4).

1.3. Chuỗi

Kiểu	Đặc điểm
`CHAR(n)`	Độ dài **cố định** n, tự đệm khoảng trắng (mã cố định như `'HNX'`)
`VARCHAR(n)`	Độ dài **thay đổi** tối đa n (tên, email, mã khách hàng)
`TEXT` (MySQL) / `VARCHAR(MAX)` (SQL Server)	Văn bản dài (ghi chú)
`NVARCHAR(n)` (SQL Server)	Chuỗi **Unicode** (tiếng Việt có dấu). MySQL dùng `VARCHAR` với bộ ký tự `utf8mb4`

Dữ liệu mẫu của khóa **không dấu** (Ha Noi, Nguyen Van An) để cùng chạy đúng ở mọi cấu hình, nhưng khi làm việc thật với tiếng Việt có dấu trong SQL Server, hằng chuỗi phải viết dạng `N'Nguyễn Văn An'` (tiền tố `N`), nếu không dấu bị mất.

Kiểu chuỗi ngắn (`customer_code` `NVARCHAR(10)`, `symbol` `NVARCHAR(10)`) và **ràng buộc** `UNIQUE` trên mã khách hàng cho thấy: mã định danh nên là chuỗi có độ dài chuẩn hóa, không phải số, vì có thể chứa tiền tố (`KH`) và số 0 đứng đầu (`KH000123`).

1.4. Ngày giờ

Kiểu	Nội dung	Ghi chú
`DATE`	Chỉ ngày `yyyy-mm-dd`	`created_date`, `trade_date`, `birth_date` — dùng kiểu này cho bài chấm
`TIME`	Chỉ giờ	
`DATETIME`	Ngày + giờ	SQL Server: độ chính xác ~3ms; MySQL: đến giây
`TIMESTAMP`	MySQL: thời điểm, tự đổi theo múi giờ phiên làm việc, giới hạn đến năm 2038. SQL Server: kiểu `timestamp` là bộ đếm phiên bản dòng, **không** phải ngày giờ (dùng `DATETIME2`)	Cẩn thận khi chuyển hệ

Cột `DATE` so sánh trực tiếp với chuỗi dạng `'yyyy-mm-dd'` và in ra đúng định dạng đó ở cả hai hệ:

```sql
SELECT customer_id, created_date
FROM Customers
WHERE created_date >= '2026-09-01'
ORDER BY customer_id;
```

```
5   2026-09-23
12  2026-09-19
19  2026-09-14
26  2026-09-23
```

`AuditLog.changed_at` là `DATETIME` có `DEFAULT GETDATE()` (MySQL: `CURRENT_TIMESTAMP`) — ta sẽ dùng nó ở phần trigger. Trong bài chấm tự động, chỉ dùng cột `DATE`, vì thời điểm hiện tại luôn thay đổi.

1.5. Boolean

- **MySQL:** có `BOOLEAN` (thực chất là `TINYINT(1)`), với hằng `TRUE` / `FALSE` = `1` / `0`, và có thể `SELECT` trực tiếp một phép so sánh.
- **SQL Server: không có** kiểu boolean thật; dùng `BIT` (giá trị `0`, `1`, `NULL`). Điều kiện không phải biểu thức giá trị nên phải bọc trong `CASE WHEN`.

MySQL:

```sql
SELECT TRUE AS dung, FALSE AS sai, 5 > 3 AS so_sanh;
```

```
1   0   1
```

SQL Server:

```sql
SELECT CAST(1 AS BIT) AS cot_bit,
       CASE WHEN 5 > 3 THEN 1 ELSE 0 END AS so_sanh;
```

```
1   1
```

Cách viết chung cho hai hệ: dùng `TINYINT` / `BIT` với `1` / `0` và luôn so sánh tường minh (`WHERE is_active = 1`).

1.6. Chuyển kiểu bằng `CAST`

`CAST(biểu_thức AS kiểu)` chuyển giá trị sang kiểu khác. Tên kiểu **đích** khác nhau giữa hai hệ:

Mục đích	SQL Server	MySQL
Sang số nguyên	`CAST(x AS INT)`	`CAST(x AS SIGNED)` — `AS INT` **báo lỗi**
Sang số thập phân	`CAST(x AS DECIMAL(18,2))`	`CAST(x AS DECIMAL(18,2))`
Sang chuỗi	`CAST(x AS VARCHAR(10))`	`CAST(x AS CHAR(10))`
Sang ngày	`CAST(x AS DATE)`	`CAST(x AS DATE)`
Dạng khác của `CAST`	`CONVERT(INT, x)`	`CONVERT(x, SIGNED)`

`DECIMAL(18,2)` và `DATE` dùng chung được ở cả hai hệ nên là lựa chọn ưu tiên khi viết SQL "đa nền tảng".

SQL Server — ép số thập phân sang số nguyên bằng `INT` thì **cắt bỏ** phần lẻ:

```
SELECT CAST(95680.50 AS DECIMAL(10,0)) AS sang_decimal,
       CAST(27270.50 AS INT) AS sang_int;
```

```
95681   27270
```

MySQL — ép sang `SIGNED` thì **làm tròn**:

```
SELECT CAST(95680.50 AS DECIMAL(10,0)) AS sang_decimal,
       CAST(27270.50 AS SIGNED) AS sang_signed;
```

```
95681   27271
```

> **SQL Server vs MySQL:** `CAST(27270.50 AS INT)` cho **27270** ở SQL Server nhưng `CAST(27270.50 AS SIGNED)` cho **27271** ở MySQL. Muốn kết quả giống nhau ở cả hai hệ, làm tròn tường minh: `CAST(ROUND(x, 0) AS DECIMAL(18,0))` .

`CAST` cũng dùng để biến chuỗi thành số, ví dụ lấy phần số của mã khách hàng (`KH100007` → 100007):

SQL Server:

```
SELECT customer_id, CAST(SUBSTRING(customer_code, 3, 6) AS INT) AS so_ma
FROM Customers WHERE customer_id IN (1, 2) ORDER BY customer_id;
```

```
1   100007
2   100014
```

MySQL:

```
SELECT customer_id, CAST(SUBSTRING(customer_code, 3, 6) AS SIGNED) AS so_ma
FROM Customers WHERE customer_id IN (1, 2) ORDER BY customer_id;
```

```
1   100007
2   100014
```

> **Cảnh báo: Chuyển kiểu ngầm** (implicit conversion) xảy ra khi so sánh cột chuỗi với hằng số, ví dụ `WHERE customer_code = 100007`. SQL Server cố đổi **cả cột** `'KH100007'` sang số và báo lỗi `Conversion failed when converting the nvarchar value 'KH100007' to data type int`. MySQL lặng lẽ đổi chuỗi thành số (`'KH100007'` thành 0) nên câu lệnh **không lỗi nhưng trả về rỗng** — dễ bị bỏ sót hơn. Luôn so sánh cùng kiểu: `WHERE customer_code = 'KH100007'`.

1.7. Chọn kiểu nào cho dữ liệu chứng khoán?

Dữ liệu	Kiểu nên dùng	Lý do
Giá cổ phiếu (`close_price`)	`DECIMAL(12,2)`	Chính xác đến đồng, không sai số
Số dư, giá trị giao dịch	`DECIMAL(18,2)` / `DECIMAL(20,2)`	Đủ lớn cho hàng nghìn tỷ đồng
Tỷ lệ phí (0,05%)	`DECIMAL(6,4)`	Giữ đúng 4 chữ số thập phân
Khối lượng khớp (`volume`)	`BIGINT`	Tổng cả thị trường có thể vượt 2,1 tỷ
Số lượng trong một lệnh	`INT`	Đủ dùng, tiết kiệm
Mã chứng khoán, mã khách hàng	`VARCHAR(10)` / `NVARCHAR(10)`	Có chữ và số 0 đứng đầu
Ngày giao dịch, ngày sinh	`DATE`	Không cần giờ
Cờ đúng/sai (đã xác thực?)	`BIT` (SQL Server) / `TINYINT(1)` (MySQL)	Không có boolean thật ở SQL Server

> **Ví dụ nghiệp vụ:** Nếu lưu `close_price` bằng `FLOAT`, giá trị danh mục 10 triệu dòng có thể lệch vài đồng so với sổ sách của bộ phận kế toán — và lệch này không tái lập được. Với `DECIMAL(12,2)` số liệu luôn khớp.

Sai lầm thường gặp: dùng `FLOAT` cho tiền; để chuỗi so sánh với số (chuyển kiểu ngầm, ở trên); quên rằng `INT * INT` có thể **tràn** (`quantity * price` nếu cả hai là `INT` lớn) — hãy ép một vế sang `BIGINT` / `DECIMAL` trước khi nhân.

---

2. Hàm xử lý chuỗi

Ví dụ nền: `customer_code` dạng `KH100007` (2 chữ + 6 số), `full_name` như `Nguyen Van An`, `email` như `nguyen.van.an@example.com`.

2.1. Độ dài, cắt, nối, đổi hoa/thường

Nhu cầu	SQL Server	MySQL	Ví dụ → kết quả
Độ dài (số ký tự)	`LEN(s)`	`CHAR_LENGTH(s)`	`'Nguyen Van An'` → 13
Cắt N ký tự cuối / đầu	`RIGHT(s, n)` / `LEFT(s, n)`	giống	`RIGHT('KH100007', 3)` → `007`
Lấy đoạn giữa	`SUBSTRING(s, bắt_đầu, độ_dài)`	`SUBSTRING(s, bắt_đầu[, độ_dài])`	`SUBSTRING('KH100007', 3, 6)` → `100007`
Nối chuỗi	`CONCAT(a, b, ...)` hoặc `a + b`	`CONCAT(a, b, ...)`	
Chữ hoa / thường	`UPPER(s)` / `LOWER(s)`	giống	
Cắt khoảng trắng đầu-cuối	`TRIM(s)` (từ 2017), hoặc `LTRIM(RTRIM(s))`	`TRIM(s)`	`TRIM(' abc ')` → `abc`
Thay thế	`REPLACE(s, cũ, mới)`	giống	
Vị trí ký tự (đếm từ 1, **0 = không có**)	`CHARINDEX(cần_tìm, s)`	`INSTR(s, cần_tìm)` (hoặc `LOCATE(cần_tìm, s)`)	`INSTR('Nguyen Van An', ' ')` → 7

Kết quả thực tế trên bộ dữ liệu (chạy giống nhau ở cả hai hệ):

```sql
SELECT customer_id, customer_code,
       RIGHT(customer_code, 3) AS r3,
       SUBSTRING(customer_code, 3, 6) AS so,
       UPPER(full_name) AS hoa,
       LOWER(full_name) AS thuong
FROM Customers WHERE customer_id IN (1, 6, 31)
ORDER BY customer_id;
```

```
1   KH100007    007 100007  NGUYEN VAN AN   nguyen van an
6   KH100042    042 100042  VU THI HOA  vu thi hoa
31  KH100217    217 100217  NGAN HANG TMCP DAU TU VIET  ngan hang tmcp dau tu viet
```

`TRIM`, `LTRIM`, `RTRIM` bỏ khoảng trắng thừa (dữ liệu nhập tay thường có) và `LEFT` lấy các ký tự đầu:

```sql
SELECT TRIM('  KH100007  ') AS da_trim,
       LTRIM('  abc') AS cat_trai,
       RTRIM('abc  ') AS cat_phai,
       LEFT('KH100007', 2) AS tien_to;
```

```
KH100007    abc abc KH
```

`REPLACE` dùng để chuẩn hóa tên công ty:

```
SELECT customer_id, REPLACE(full_name, 'Cong ty CP', 'CTCP') AS ten_gon
FROM Customers WHERE customer_id IN (35, 38)
ORDER BY customer_id;
```

```
35  CTCP Chung khoan An Binh
38  CTCP Xay dung Song Hong
```

Ghép `LEFT`, `RIGHT` và `CONCAT` để **che số điện thoại** khi in báo cáo:

```
SELECT customer_id, CONCAT(LEFT(phone, 4), '****', RIGHT(phone, 3)) AS sdt_che
FROM Customers WHERE customer_id IN (1, 2, 3)
ORDER BY customer_id;
```

```
1   0910****137
2   0910****274
3   0910****411
```

2.2. Ba khác biệt SQL Server vs MySQL cần thuộc

(1) `LEN` khác `CHAR_LENGTH` khi có khoảng trắng cuối.

SQL Server — `LEN` bỏ qua khoảng trắng cuối:

```
SELECT LEN('KH100007  ') AS do_dai;
```

```
8
```

MySQL — `CHAR_LENGTH` đếm cả khoảng trắng; `LENGTH` đếm **byte**, không phải ký tự:

```
SELECT CHAR_LENGTH('KH100007  ') AS char_length, LENGTH('KH100007') AS length_byte;
```

```
10  8
```

Với tiếng Việt có dấu trong MySQL, `LENGTH('Đà Nẵng')` lớn hơn số ký tự thật; luôn dùng `CHAR_LENGTH`. Cách chung: `TRIM` trước khi đếm.

(2) `CONCAT` xử lý `NULL` khác nhau. Cột `email` của khách 6 là `NULL`:

SQL Server — coi `NULL` là chuỗi rỗng:

```sql
SELECT customer_id, CONCAT(full_name, ' <', email, '>') AS voi_email
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   Hoang Van Em <hoang.van.em@example.com>
6   Vu Thi Hoa <>
```

MySQL — trả về `NULL` nếu **bất kỳ** đối số nào là `NULL`:

```sql
SELECT customer_id, CONCAT(full_name, ' <', email, '>') AS voi_email
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   Hoang Van Em <hoang.van.em@example.com>
6   NULL
```

Cách an toàn cho cả hai: `CONCAT(full_name, ' <', COALESCE(email, ''), '>')`.

```sql
SELECT customer_id, CONCAT(full_name, ' <', COALESCE(email, ''), '>') AS an_toan
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   Hoang Van Em <hoang.van.em@example.com>
6   Vu Thi Hoa <>
```

(3) Vị trí chuỗi con. T-SQL `CHARINDEX(cần_tìm, s)` — thứ tự tham số **ngược** với MySQL `INSTR(s, cần_tìm)`. Cả hai đều trả `0` nếu không tìm thấy.

SQL Server:

```sql
SELECT CHARINDEX(' ', 'Nguyen Van An') AS co_khoang_trang, CHARINDEX('@', 'abc') AS
        khong_tim_thay;
```

```
7   0
```

MySQL:

```sql
SELECT INSTR('Nguyen Van An', ' ') AS co_khoang_trang, INSTR('abc', '@') AS khong_tim_thay;
```

```
7   0
```

> **SQL Server vs MySQL:** `SUBSTRING` với độ dài âm sẽ **lỗi** ở T-SQL nhưng trả chuỗi rỗng ở MySQL — nên kiểm tra `= 0` trước bằng `CASE`. Ngoài ra SQL Server bắt buộc có tham số độ dài trong `SUBSTRING`, MySQL thì không.

2.3. Ứng dụng: tách mã khách hàng, định dạng tên

Lấy phần số của mã (`KH100007` → 100007) và **3 số cuối**: `RIGHT(customer_code, 3)`. **Tách họ và phần tên** ở khoảng trắng đầu tiên:

SQL Server:

```
SELECT customer_id,
       SUBSTRING(full_name, 1, CHARINDEX(' ', full_name) - 1) AS ho,
       SUBSTRING(full_name, CHARINDEX(' ', full_name) + 1, 200) AS phan_con_lai
FROM Customers WHERE customer_id IN (1, 2, 3) ORDER BY customer_id;
```

```
1   Nguyen  Van An
2   Tran    Thi Binh
3   Le  Van Cuong
```

MySQL:

```
SELECT customer_id,
       SUBSTRING(full_name, 1, INSTR(full_name, ' ') - 1) AS ho,
       SUBSTRING(full_name, INSTR(full_name, ' ') + 1) AS phan_con_lai
FROM Customers WHERE customer_id IN (1, 2, 3) ORDER BY customer_id;
```

```
1   Nguyen  Van An
2   Tran    Thi Binh
3   Le  Van Cuong
```

Tên chỉ có một từ (không có khoảng trắng) làm `INSTR = 0` → độ dài `-1`: hãy bọc bằng `CASE WHEN ... = 0 THEN ... ELSE ... END`.

Tách tên miền email (phần sau `@`), an toàn với `NULL` nhờ hàm của SQL đã trả `NULL` khi đầu vào `NULL`:

SQL Server:

```
SELECT customer_id, SUBSTRING(email, CHARINDEX('@', email) + 1, 100) AS mien
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   example.com
6   NULL
```

MySQL:

```
SELECT customer_id, SUBSTRING(email, INSTR(email, '@') + 1) AS mien
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   example.com
6   NULL
```

Sai lầm thường gặp: đếm độ dài bằng `LENGTH` trong MySQL (đếm byte); nối chuỗi bằng `+` trong SQL Server khi có `NULL` (cả biểu thức thành `NULL`); quên rằng `SUBSTRING` đếm vị trí **từ 1**, không phải từ 0; so sánh chuỗi mà quên rằng hoa/thường thường **không phân biệt** với collation mặc định.

Thực hành ngay: bài 4.1

Vừa học xong `RIGHT`, `UPPER`, `LEN/CHAR_LENGTH`, `TRIM` và tách chuỗi bằng `CHARINDEX/INSTR`, hãy làm nhóm **bài 4.1** để trích mã khách hàng, chuẩn hóa tên và ghép nhãn thư từ.

- **Bài 4.1** — Trích mã, chữ hoa và độ dài tên khách hàng *(độ khó: dễ)*
- **Bài 4.1** — Tách họ và ghép nhãn tên khách hàng *(độ khó: trung bình)*

---

3. Hàm xử lý ngày giờ

3.1. Ngày giờ hiện tại — và vì sao khóa học không dùng

Nhu cầu	SQL Server	MySQL
Ngày giờ hiện tại	`GETDATE()`, `SYSDATETIME()`	`NOW()`
Chỉ ngày hiện tại	`CAST(GETDATE() AS DATE)`	`CURDATE()`
Chỉ giờ hiện tại	`CAST(GETDATE() AS TIME)`	`CURTIME()`

Các hàm này trả về giá trị **thay đổi mỗi lần chạy**, nên chấm tự động không thể so kết quả cố định. Trong khóa học, hãy thay bằng ngày cố định `'2026-09-25'`. Trong hệ thống thật thì dùng hàm tương ứng ở bảng trên.

3.2. Trích thành phần của ngày

`YEAR(d)`, `MONTH(d)`, `DAY(d)` **giống nhau ở cả hai hệ**. Ngày trong tuần:

	SQL Server	MySQL
Thứ (số)	`DATEPART(weekday, d)`	`DAYOFWEEK(d)`
Tên thứ	`DATENAME(weekday, d)`	`DAYNAME(d)`
Quý	`DATEPART(quarter, d)`	`QUARTER(d)`

SQL Server:

```sql
SELECT DATENAME(weekday, '2021-06-29') AS ten_thu,
       DATEPART(weekday, '2021-06-29') AS so_thu,
       DATEPART(quarter, '2026-09-25') AS quy,
       EOMONTH('2026-09-25') AS cuoi_thang;
```

```
Tuesday 3   3   2026-09-30
```

MySQL:

```sql
SELECT DAYNAME('2021-06-29') AS ten_thu,
       DAYOFWEEK('2021-06-29') AS so_thu,
       QUARTER('2026-09-25') AS quy,
       LAST_DAY('2026-09-25') AS cuoi_thang;
```

```
Tuesday 3   3   2026-09-30
```

Với cấu hình mặc định của khóa, Chủ nhật = 1, thứ Hai = 2, ...; `'2021-06-29'` là **Tuesday** (3).

> **Lưu ý:** Ở SQL Server kết quả `DATEPART(weekday, ...)` phụ thuộc thiết lập `DATEFIRST` của phiên — không nên dựa vào số thứ tự trong báo cáo quan trọng; dùng `DATENAME` / `DAYNAME` hoặc đặt rõ `SET DATEFIRST` .

3.3. Cộng/trừ ngày và chênh lệch ngày

Nhu cầu	SQL Server	MySQL
Cộng ngày	`DATEADD(day, 30, d)`	`DATE_ADD(d, INTERVAL 30 DAY)`
Trừ ngày	`DATEADD(day, -30, d)`	`DATE_SUB(d, INTERVAL 30 DAY)`
Cộng tháng/năm	`DATEADD(month, 1, d)` , `DATEADD(year, 1, d)`	`DATE_ADD(d, INTERVAL 1 MONTH)`
Số ngày giữa hai ngày	`DATEDIFF(day, ngày_đầu, ngày_cuối)`	`DATEDIFF(ngày_cuối, ngày_đầu)`
Chênh lệch theo tháng/năm	`DATEDIFF(month, đầu, cuối)`	`TIMESTAMPDIFF(MONTH, đầu, cuối)`
Cuối tháng	`EOMONTH(d)`	`LAST_DAY(d)`

> **Cảnh báo:** Thứ tự tham số `DATEDIFF` của hai hệ **ngược nhau**, và MySQL `DATEDIFF` chỉ nhận 2 tham số (đơn vị là ngày). Ví dụ số ngày khách hàng đã tham gia:

SQL Server:

```sql
SELECT customer_id, created_date,
       DATEDIFF(day, created_date, '2026-09-25') AS so_ngay,
       DATEADD(day, 30, created_date) AS sau_30_ngay
FROM Customers WHERE customer_id IN (1, 5) ORDER BY customer_id;
```

```
1   2021-06-29  1914    2021-07-29
5   2026-09-23  2   2026-10-23
```

MySQL:

```sql
SELECT customer_id, created_date,
       DATEDIFF('2026-09-25', created_date) AS so_ngay,
       DATE_ADD(created_date, INTERVAL 30 DAY) AS sau_30_ngay
FROM Customers WHERE customer_id IN (1, 5) ORDER BY customer_id;
```

```
1   2021-06-29  1914    2021-07-29
5   2026-09-23  2   2026-10-23
```

Cộng tháng và lấy ngày cuối tháng cho ngày đăng ký (dùng khi tính kỳ tính phí):

SQL Server:

```sql
SELECT customer_id, DATEADD(month, 1, created_date) AS thang_sau,
       EOMONTH(created_date) AS cuoi_thang
FROM Customers WHERE customer_id IN (1, 5) ORDER BY customer_id;
```

```
1   2021-07-29  2021-06-30
5   2026-10-23  2026-09-30
```

MySQL:

```sql
SELECT customer_id, DATE_ADD(created_date, INTERVAL 1 MONTH) AS thang_sau,
       LAST_DAY(created_date) AS cuoi_thang
FROM Customers WHERE customer_id IN (1, 5) ORDER BY customer_id;
```

```
1   2021-07-29  2021-06-30
5   2026-10-23  2026-09-30
```

Bẫy kiểu dữ liệu trả về của `DATEADD`: khi đối số là **chuỗi** (`'2026-09-25'`), T-SQL `DATEADD` trả về `DATETIME`. Cột kiểu `DATE` thì trả về `DATE`. Muốn in đúng ngày, bọc `CAST(... AS DATE)`:

SQL Server:

```sql
SELECT DATEADD(day, -30, '2026-09-25') AS tra_datetime,
       CAST(DATEADD(day, -30, '2026-09-25') AS DATE) AS tra_date;
```

```
2026-08-26 00:00:00.000 2026-08-26
```

MySQL:

```sql
SELECT DATE_SUB('2026-09-25', INTERVAL 30 DAY) AS ba_muoi_ngay_truoc;
```

```
2026-08-26
```

Bẫy `DATEDIFF(year, ...)` của SQL Server: nó đếm **số mốc 31/12–1/1 đã đi qua**, không phải số năm tròn. Hai ngày chỉ cách nhau một ngày vẫn cho 1 năm chênh lệch:

SQL Server:

```sql
SELECT DATEDIFF(year, '2025-12-31', '2026-01-01') AS so_nam,
       DATEDIFF(month, '2026-08-31', '2026-09-01') AS so_thang;
```

```
1   1
```

MySQL — `TIMESTAMPDIFF` chỉ đếm các đơn vị **tròn**:

```sql
SELECT TIMESTAMPDIFF(YEAR, '2025-12-31', '2026-01-01') AS so_nam,
       TIMESTAMPDIFF(MONTH, '2026-08-31', '2026-09-01') AS so_thang;
```

```
0   0
```

Do đó, **tính tuổi** phải xét cả tháng-ngày. Cách sau chạy được ở cả hai hệ (tuổi tại ngày 2026-09-25):

```sql
SELECT customer_id, birth_date,
       2026 - YEAR(birth_date)
         - CASE WHEN MONTH(birth_date) * 100 + DAY(birth_date) > 925 THEN 1 ELSE 0 END AS
        tuoi
FROM Customers
WHERE customer_id IN (1, 2, 3, 4, 5, 6)
ORDER BY customer_id;
```

```
1   1994-07-21  32
2   1997-12-24  28
3   1968-10-23  57
4   1994-10-14  31
5   1996-02-06  30
6   1969-07-22  57
```

Trong đó `MONTH * 100 + DAY` biến ngày sinh thành số `mmdd` (ví dụ 21/07 → `721`) để so sánh với `925` (25/09): nếu sinh nhật chưa tới thì trừ 1 tuổi.

3.4. Định dạng hiển thị ngày

	SQL Server	MySQL
Định dạng	`FORMAT(d, 'dd/MM/yyyy')`	`DATE_FORMAT(d, '%d/%m/%Y')`
Tháng dạng `yyyy-MM`	`FORMAT(d, 'yyyy-MM')`	`DATE_FORMAT(d, '%Y-%m')`

SQL Server:

```sql
SELECT FORMAT(CAST('2021-06-29' AS DATE), 'dd/MM/yyyy') AS ngay_vn,
       FORMAT(CAST('2021-06-29' AS DATE), 'yyyy-MM') AS thang;
```

```
29/06/2021  2021-06
```

MySQL:

```sql
SELECT DATE_FORMAT('2021-06-29', '%d/%m/%Y') AS ngay_vn,
       DATE_FORMAT('2021-06-29', '%Y-%m') AS thang;
```

```
29/06/2021  2021-06
```

> **Lưu ý:** Mẫu định dạng của SQL Server phân biệt hoa/thường (`MM` = tháng, `mm` = phút); của MySQL dùng `%Y %m %d`. `FORMAT` trả về **chuỗi** — dùng cho hiển thị, không dùng để so sánh ngày.

3.5. Ứng dụng: phân tích xu hướng theo mùa/kỳ

Gom nhóm theo `YEAR` / `MONTH` /quý để xem xu hướng, ví dụ số khách đăng ký theo năm (chạy giống nhau ở cả hai hệ):

```sql
SELECT YEAR(created_date) AS nam_dk, COUNT(*) AS so_khach
FROM Customers
GROUP BY YEAR(created_date)
ORDER BY nam_dk;
```

```
2018    1
2019    3
2020    7
2021    9
2022    5
2023    5
2024    4
2025    2
2026    4
```

Theo **quý** để tìm "mùa cao điểm" mở tài khoản:

SQL Server:

```sql
SELECT DATEPART(quarter, created_date) AS quy, COUNT(*) AS so_khach
FROM Customers
GROUP BY DATEPART(quarter, created_date)
ORDER BY quy;
```

```
1   8
2   10
3   14
4   8
```

MySQL:

```sql
SELECT QUARTER(created_date) AS quy, COUNT(*) AS so_khach
FROM Customers
GROUP BY QUARTER(created_date)
ORDER BY quy;
```

```
1   8
2   10
3   14
4   8
```

> **Ví dụ nghiệp vụ:** Quý 3 có nhiều khách đăng ký nhất trong dữ liệu mẫu. Phòng marketing dùng số liệu theo quý/tháng như vậy để lập kế hoạch chiến dịch mở tài khoản.

Đăng ký "trong 30 ngày qua" (tính đến 25/09/2026): điều kiện `created_date >= (ngày hôm nay − 30 ngày) AND created_date <= hôm nay`, với `DATEADD(day, -30, '2026-09-25')` (T-SQL) / `DATE_SUB('2026-09-25', INTERVAL 30 DAY)` (MySQL). Luôn **chốt rõ biên**: 30 ngày trước là bao gồm hay không, và ngày sau "hôm nay" có bị loại không — đây chính là kiểu điều kiện biên mà bài chấm sẽ kiểm tra.

Sai lầm thường gặp: đảo thứ tự tham số `DATEDIFF` giữa hai hệ (ra số âm); dùng `DATEDIFF(year, ...)` để tính tuổi; so sánh kết quả `FORMAT` (chuỗi) với ngày; quên ép `DATEADD` với chuỗi về `DATE`.

Thực hành ngay: bài 4.2

Vừa học xong `DATEDIFF`, `DATEADD/DATE_ADD`, `YEAR/MONTH`, `FORMAT/DATE_FORMAT` và cách tính tuổi, hãy làm nhóm **bài 4.2** để tính số ngày tham gia, phân loại theo năm đăng ký, tính tuổi, lọc khách mới trong 30 ngày và tính ngày thanh toán T+2.

- **Bài 4.2** — Số ngày khách hàng đã tham gia *(độ khó: dễ)*
- **Bài 4.2** — Phân loại khách hàng theo năm đăng ký *(độ khó: dễ)*
- **Bài 4.2** — Tính tuổi khách hàng *(độ khó: trung bình)*
- **Bài 4.2** — Khách hàng đăng ký trong 30 ngày qua *(độ khó: trung bình)*
- **Bài 4.2** — Ngày thanh toán T+2 và tháng giao dịch *(độ khó: trung bình)*

---

4. Hàm toán học

Hàm	Ý nghĩa	SQL Server	MySQL
`ABS(x)`	Giá trị tuyệt đối	có	có
`ROUND(x, n)`	Làm tròn `n` chữ số thập phân (`n` âm: làm tròn hàng chục/trăm)	có (giữ nguyên số chữ số thập phân của kiểu)	có
`CEILING(x)`	Làm tròn **lên**	`CEILING`	`CEILING` hoặc `CEIL`
`FLOOR(x)`	Làm tròn **xuống**	có	có
`POWER(x, y)`	Lũy thừa	có	có (`POW`)
`SQRT(x)`	Căn bậc hai	có	có
Chia lấy dư	`a % b`	`a % b` hoặc `MOD(a, b)`	

Kết quả thô của các hàm này khác nhau ở **số chữ số thập phân hiển thị**:

SQL Server:

```
SELECT ABS(-810.5) AS a, ROUND(123.456, 2) AS r2, ROUND(1234.5, -2) AS rm2,
       CEILING(3.2) AS c, FLOOR(-3.2) AS f, SQRT(144) AS s, 17 % 5 AS m;
```

```
810.5   123.460 1200.0  4   -4  12  2
```

MySQL:

```
SELECT ABS(-810.5) AS a, ROUND(123.456, 2) AS r2, ROUND(1234.5, -2) AS rm2,
       CEILING(3.2) AS c, FLOOR(-3.2) AS f, SQRT(144) AS s, 17 % 5 AS m;
```

```
810.5   123.46  1200    4   -4  12.0    2
```

Các chữ số thập phân hiển thị khác nhau (`123.460` so với `123.46` , `12` so với `12.0`); vì vậy khi cần khớp chính xác, luôn `CAST(ROUND(x, 2) AS DECIMAL(18,2))` :

```
SELECT CAST(ABS(-810.5) AS DECIMAL(18,2)) AS a,
       CAST(ROUND(123.456, 2) AS DECIMAL(18,2)) AS r2,
       CAST(CEILING(3.2) AS DECIMAL(18,0)) AS c,
       CAST(FLOOR(-3.2) AS DECIMAL(18,0)) AS f,
       CAST(SQRT(144) AS DECIMAL(18,2)) AS s,
       17 % 5 AS m;
```

```
810.50  123.46  4   -4  12.00   2
```

Cắt bớt thay vì làm tròn. `ROUND` làm tròn theo quy tắc số học; muốn cắt bỏ phần lẻ thì T-SQL dùng tham số thứ ba `ROUND(x, n, 1)` , MySQL dùng `TRUNCATE(x, n)` :

SQL Server:

```
SELECT ROUND(123.456, 2) AS lam_tron, ROUND(123.456, 2, 1) AS cat_bo;
```

```
123.460 123.450
```

MySQL:

```
SELECT ROUND(123.456, 2) AS lam_tron, TRUNCATE(123.456, 2) AS cat_bo;
```

```
123.46  123.45
```

Làm tròn lên/xuống thế nào cho đúng nghiệp vụ. Giá trị giao dịch 718,3 tỷ quy ra "số tỷ làm tròn lên" là `CEILING(718.3)` = 719; `FLOOR(718.9)` = 718. Lô giao dịch chuẩn là 100 cổ phiếu: số lô = `FLOOR(quantity / 100.0)` và phần lẻ = `quantity % 100` :

```sql
SELECT order_id, quantity, FLOOR(quantity / 100.0) AS so_lo, quantity % 100 AS le
FROM Orders WHERE order_id <= 5 ORDER BY order_id;
```

```
1   300 3   0
2   2000    20  0
3   2000    20  0
4   100 1   0
5   1000    10  0
```

Bẫy `POWER` với kiểu `DECIMAL` của SQL Server. Hàm `POWER` trả về **cùng kiểu** với đối số thứ nhất — với `DECIMAL(3,2)` nghĩa là chỉ giữ 2 chữ số thập phân:

SQL Server:

```sql
SELECT POWER(1.06, 3) AS sai, POWER(1.06e0, 3) AS dung;
```

```
1.19    1.191016
```

MySQL:

```sql
SELECT POWER(1.06, 3) AS sai, POWER(1.06e0, 3) AS dung;
```

```
1.191016    1.191016
```

> **Cảnh báo:** Ở SQL Server, `POWER(1.06, 3)` bị cắt còn `1.19` (sai). Cách xử lý: ép đối số sang **số thực** bằng hằng `1.06e0` (hoặc nhân `1.0e0`), rồi làm tròn/ép kiểu ở bước cuối.

Ví dụ dự phóng số dư 8.000.000 đồng gửi 3 năm, lãi kép 6%/năm — cho cùng kết quả ở cả hai hệ:

```sql
SELECT CAST(ROUND(8000000 * POWER(1.06e0, 3), 2) AS DECIMAL(18,2)) AS so_du_sau_3_nam;
```

```
9528128.00
```

Ứng dụng: làm tròn lợi suất, tính phần trăm. Lợi suất trong ngày = `100.0 * (close_price - open_price) / open_price`; nhớ `100.0` (thập phân) để tránh chia nguyên, rồi `ROUND(..., 2)`:

```sql
SELECT symbol, open_price, close_price,
       CAST(ROUND(100.0 * (close_price - open_price) / open_price, 2) AS DECIMAL(18,2)) AS
        loi_suat_pct
FROM MarketPrices
WHERE trade_date = '2026-09-25' AND symbol IN ('FPT', 'HPG', 'VCB')
ORDER BY symbol;
```

```
FPT 118800.00   116400.00   -2.02
HPG 26460.00    27270.00    3.06
VCB 92920.00    95680.00    2.97
```

Độ lệch chuẩn (đo rủi ro) = `SQRT(AVG(POWER(x - trung_bình, 2)))` — cần ép sang số thực như trên.

Sai lầm thường gặp: quên `100.0` khiến phần trăm bị chia nguyên; `POWER` trên `DECIMAL` ở SQL Server; hiển thị `ROUND` thô rồi so sánh chuỗi với đáp án (số chữ số khác nhau giữa hai hệ).

Thực hành ngay: bài 4.4

Vừa học xong `ROUND`, `ABS`, `CEILING`, `POWER`, `SQRT`, hãy làm nhóm **bài 4.4** để tính lợi suất trong ngày, biên độ, giá trị làm tròn lên tỷ đồng và độ lệch chuẩn giá của từng mã.

- **Bài 4.4** — Lợi suất trong ngày, biên độ và giá trị làm tròn lên tỷ *(độ khó: trung bình)*
- **Bài 4.4** — Độ biến động giá (độ lệch chuẩn) của từng mã *(độ khó: khó)*

---

5. Hàm điều kiện

5.1. `CASE WHEN` — chuẩn SQL, chạy trên cả hai hệ

```sql
CASE
    WHEN điều_kiện_1 THEN giá_trị_1
    WHEN điều_kiện_2 THEN giá_trị_2
    ELSE giá_trị_mặc_định
END
```

Các nhánh được xét **từ trên xuống, nhánh đúng đầu tiên thắng**; không có `ELSE` và không nhánh nào đúng thì kết quả là `NULL`. Ví dụ phân nhóm mức giá:

```sql
SELECT symbol, close_price,
       CASE WHEN close_price > 100000 THEN 'Cao'
            WHEN close_price > 30000  THEN 'Trung binh'
            ELSE 'Thap' END AS nhom_gia
FROM MarketPrices
WHERE trade_date = '2026-09-25' AND symbol IN ('FPT', 'VNM', 'SHS')
ORDER BY symbol;
```

```
FPT 116400.00   Cao
SHS 14720.00    Thap
VNM 65960.00    Trung binh
```

`CASE` dùng được **ở nhiều nơi**: trong `SELECT`, `ORDER BY`, `WHERE`, và bên trong hàm tổng hợp để **đếm có điều kiện**:

```sql
SELECT SUM(CASE WHEN status = 'ACTIVE' THEN 1 ELSE 0 END) AS dang_hoat_dong,
       SUM(CASE WHEN status <> 'ACTIVE' THEN 1 ELSE 0 END) AS khac
FROM Customers;
```

```
31  9
```

Khi `GROUP BY` một biểu thức `CASE`, phải **lặp lại nguyên biểu thức** (hoặc gom ở truy vấn con):

```sql
SELECT CASE WHEN side = 'BUY' THEN 'Mua'
            WHEN side = 'SELL' THEN 'Ban'
            ELSE 'Khac' END AS loai,
       COUNT(*) AS so_lenh
FROM Orders
GROUP BY CASE WHEN side = 'BUY' THEN 'Mua'
              WHEN side = 'SELL' THEN 'Ban'
              ELSE 'Khac' END
ORDER BY loai;
```

```
Ban 42
Mua 78
```

> **Cảnh báo:** Nhánh `WHEN` hẹp đặt sau nhánh rộng sẽ bị "nuốt": `WHEN close_price > 30000` đặt trước `WHEN close_price > 100000` thì mọi giá trên 100.000 đều rơi vào nhánh đầu. Luôn xếp từ điều kiện **hẹp/chặt nhất** đến rộng nhất.

Hình dưới minh họa cách `CASE WHEN` xét điều kiện lần lượt từ trên xuống — cơ chế quyết định thứ tự viết các nhánh.

Hình minh họa: Luồng chạy của CASE WHEN (xem hình động trên trang bài giảng trực tuyến).

5.2. `IF` / `IIF`

Nhánh hai lựa chọn ngắn gọn:

SQL Server	MySQL
`IIF(điều_kiện, giá_trị_đúng, giá_trị_sai)` (từ 2012)	`IF(điều_kiện, giá_trị_đúng, giá_trị_sai)`

SQL Server:

```
SELECT symbol, open_price, close_price,
       IIF(close_price >= open_price, 'Tang', 'Giam') AS chieu
FROM MarketPrices
WHERE trade_date = '2026-09-25' AND symbol IN ('FPT', 'HPG', 'VCB')
ORDER BY symbol;
```

```
FPT 118800.00   116400.00   Giam
HPG 26460.00    27270.00    Tang
VCB 92920.00    95680.00    Tang
```

MySQL:

```
SELECT symbol, open_price, close_price,
       IF(close_price >= open_price, 'Tang', 'Giam') AS chieu
FROM MarketPrices
WHERE trade_date = '2026-09-25' AND symbol IN ('FPT', 'HPG', 'VCB')
ORDER BY symbol;
```

```
FPT 118800.00   116400.00   Giam
HPG 26460.00    27270.00    Tang
VCB 92920.00    95680.00    Tang
```

> **SQL Server vs MySQL:** `IF(...)` chỉ chạy trên MySQL; `IIF(...)` chỉ chạy trên SQL Server. **Muốn một câu lệnh chạy cả hai hệ thì dùng `CASE`.** (Trong stored procedure, `IF ... THEN ... END IF` của MySQL và `IF ... ELSE` của T-SQL là *câu lệnh điều khiển luồng*, khác với hàm `IF()` ở đây — sẽ học ở Buổi 9.)

5.3. `COALESCE`, `IFNULL` / `ISNULL`, `NULLIF`

Hàm	Ý nghĩa	Dùng ở
`COALESCE(a, b, c, ...)`	Trả về giá trị **không NULL đầu tiên**	Cả hai hệ (chuẩn SQL)
`IFNULL(a, b)`	Nếu `a` là NULL thì `b`	MySQL
`ISNULL(a, b)`	Nếu `a` là NULL thì `b`	SQL Server (MySQL cũng có hàm `ISNULL(x)` nhưng **ý nghĩa khác**: kiểm tra NULL, trả 0/1)
`NULLIF(a, b)`	Trả `NULL` nếu `a = b`, ngược lại trả `a`	Cả hai — thường dùng để tránh chia cho 0: `x / NULLIF(y, 0)`

```sql
SELECT customer_id, email, COALESCE(email, 'chua co') AS email_hien_thi
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   hoang.van.em@example.com    hoang.van.em@example.com
6   NULL    chua co
```

Cùng kết quả bằng hàm riêng của từng hệ:

SQL Server:

```sql
SELECT customer_id, ISNULL(email, 'chua co') AS email_hien_thi
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   hoang.van.em@example.com
6   chua co
```

MySQL:

```sql
SELECT customer_id, IFNULL(email, 'chua co') AS email_hien_thi
FROM Customers WHERE customer_id IN (5, 6) ORDER BY customer_id;
```

```
5   hoang.van.em@example.com
6   chua co
```

Đếm dữ liệu thiếu bằng `CASE` (cách này không phát sinh cảnh báo "NULL bị loại" của SQL Server như `COUNT(cột)`):

```sql
SELECT COUNT(*) AS so_kh,
       SUM(CASE WHEN email IS NULL THEN 1 ELSE 0 END) AS thieu_email
FROM Customers;
```

```
40  5
```

Ứng dụng quan trọng: **tổng có thể là NULL**. Khách chưa mở tài khoản nào thì `SUM(balance)` là `NULL`, và `NULL > 1000000000` không đúng cũng không sai → khách rơi khỏi cả hai nhánh `CASE`. Bọc `COALESCE(SUM(...), 0)` để coi là 0.

5.4. Ba ứng dụng nghiệp vụ điển hình (nền tảng của bài 4.3)

Phân loại khách hàng VIP/Thường theo tổng số dư mọi tài khoản. Khi chưa học `JOIN`, ta lấy tổng bằng **truy vấn con tương quan** (mỗi dòng khách hàng "chạy" một truy vấn nhỏ lấy tổng số dư của chính nó):

```sql
SELECT c.customer_id,
       COALESCE((SELECT SUM(a.balance) FROM Accounts a WHERE a.customer_id = c.customer_id),
        0) AS tong_tai_san
FROM Customers c
WHERE c.customer_id <= 8
ORDER BY c.customer_id;
```

```
1   67000000.00
2   8640000000.00
3   168000000.00
4   1480000000.00
5   3432000000.00
6   1600000000.00
7   1290000000.00
8   760000000.00
```

rồi bọc lại thành truy vấn con và dùng `CASE WHEN tong_tai_san > 1000000000 THEN 'VIP' ELSE 'Thuong' END`.

Đổi nhãn Mua/Bán/Khác cho cột `side` (BUY/SELL/giá trị khác) — `CASE WHEN side = 'BUY' THEN 'Mua' WHEN side = 'SELL' THEN 'Ban' ELSE 'Khac' END` (như ví dụ `GROUP BY` ở mục 5.1).

Hoa hồng khác nhau cho khách mới/cũ: tỷ lệ = `CASE WHEN created_date >= '2026-08-26' THEN 0.001 ELSE 0.0005 END`, hoa hồng = tổng giá trị giao dịch × tỷ lệ, rồi `ROUND` + `CAST DECIMAL`. Nên ép kiểu tỷ lệ để hai hệ in giống nhau:

```sql
SELECT customer_id, created_date,
       CAST(CASE WHEN created_date >= '2026-08-26' THEN 0.001 ELSE 0.0005 END AS
        DECIMAL(6,4)) AS ty_le
FROM Customers
WHERE customer_id IN (1, 5, 12)
ORDER BY customer_id;
```

```
1   2021-06-29  0.0005
5   2026-09-23  0.0010
12  2026-09-19  0.0010
```

> **Ví dụ nghiệp vụ:** Bộ phận kinh doanh áp biểu phí 0,1% cho khách đăng ký trong 30 ngày gần nhất và 0,05% cho khách cũ. Thay vì hai truy vấn riêng, một biểu thức `CASE` cho phép tính phí cho mọi khách trong một lần chạy.

Sai lầm thường gặp: đặt nhánh `WHEN` hẹp sau nhánh rộng (nhánh rộng "nuốt" trước); quên `ELSE` khiến ra `NULL`; so sánh với `NULL` bằng `=` (phải dùng `IS NULL`); trộn kiểu giá trị giữa các nhánh (một nhánh số, một nhánh chuỗi) gây lỗi chuyển kiểu.

Thực hành ngay: bài 4.3

Vừa học xong `CASE`, `COALESCE` và các ứng dụng phân loại, hãy làm nhóm **bài 4.3** để gắn nhãn Mua/Bán/Khác cho lệnh, phân loại khách VIP/Thường theo tài sản và tính hoa hồng theo khách mới/cũ.

- **Bài 4.3** — Phân loại lệnh Mua / Bán / Khác *(độ khó: dễ)*
- **Bài 4.3** — Phân loại khách hàng VIP / Thường theo tài sản *(độ khó: khó)*
- **Bài 4.3** — Tính hoa hồng theo khách hàng mới / cũ *(độ khó: khó)*

---

6. Tổng kết

Nhóm	Hàm chính	Khác biệt SQL Server ↔ MySQL
Kiểu dữ liệu	`INT`, `BIGINT`, `DECIMAL`, `VARCHAR`, `DATE`, `BIT` / `BOOLEAN`	`/` số nguyên; `CAST AS INT` ↔ `AS SIGNED` (cắt ↔ làm tròn); `VARCHAR` ↔ `CHAR` khi ép sang chuỗi
Chuỗi	`LEN/CHAR_LENGTH`, `SUBSTRING`, `RIGHT`, `UPPER`, `TRIM`, `REPLACE`, `CONCAT`	`CHARINDEX` ↔ `INSTR` (ngược tham số); `CONCAT` với NULL; `LENGTH` đếm byte
Ngày giờ	`YEAR/MONTH/DAY`, `DATEDIFF`, `DATEADD`, `FORMAT`	`DATEDIFF(day, a, b)` ↔ `DATEDIFF(b, a)`; `DATEADD` ↔ `DATE_ADD` / `DATE_SUB`; `FORMAT` ↔ `DATE_FORMAT`; `DATEDIFF(year)` sai tuổi
Toán học	`ABS`, `ROUND`, `CEILING`, `FLOOR`, `POWER`, `SQRT`, `%`	`POWER` trên `DECIMAL`; số chữ số thập phân hiển thị; cắt bớt: `ROUND(x,n,1)` ↔ `TRUNCATE`
Điều kiện	`CASE`, `COALESCE`, `NULLIF`	`IIF` ↔ `IF`; `ISNULL` ↔ `IFNULL`

Nguyên tắc vàng khi viết SQL chạy được ở cả hai hệ: dùng `CASE`, `COALESCE`, `CAST(... AS DECIMAL(18,2))`, `CONCAT` (kèm `COALESCE`), `YEAR/MONTH/DAY`; khi buộc phải khác nhau (đếm độ dài, chênh lệch ngày, tìm vị trí, cộng ngày) thì tra bảng ở các mục trên.

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Chia hai số nguyên (`a / b`)	SQL Server cắt phần thập phân, MySQL không	`100.0 * a / b` rồi `CAST(ROUND(...) AS DECIMAL(18,2))`
Lưu tiền bằng `FLOAT`	Sai lệch tích lũy, không khớp sổ sách	Dùng `DECIMAL`
`CAST(x AS INT)` trên MySQL	Lỗi cú pháp	`CAST(x AS SIGNED)` ; nhớ SQL Server cắt còn MySQL làm tròn
So sánh cột chuỗi với số	SQL Server báo lỗi; MySQL trả rỗng lặng lẽ	So sánh cùng kiểu: `= 'KH100007'`
Đếm độ dài bằng `LENGTH` ở MySQL	Đếm byte, sai với tiếng Việt có dấu	`CHAR_LENGTH`
`CONCAT` với cột có `NULL`	MySQL trả `NULL` cả chuỗi	`CONCAT(..., COALESCE(cột, ''), ...)`
`DATEDIFF` đảo tham số / `DATEDIFF(year)` tính tuổi	Ra số âm hoặc tuổi sai	Đúng thứ tự của từng hệ; tuổi = năm trừ năm, trừ 1 nếu chưa tới sinh nhật
`POWER` trên `DECIMAL` ở SQL Server	Kết quả bị cắt chữ số thập phân	Dùng hằng `1.06e0` (số thực)
`CASE` thiếu `ELSE` hoặc xếp nhánh sai thứ tự	Ra `NULL` hoặc nhãn sai	Thêm `ELSE` , xếp nhánh hẹp trước

Bài tập thực hành của buổi

Các bài đều chấm tự động trên **cả SQL Server và MySQL** (mỗi bài có lời giải riêng cho từng hệ ở chỗ cú pháp khác nhau). Nếu bạn chưa làm hết các nhóm bài đã gắn trong bài giảng, đây là danh sách đầy đủ của buổi.

- **Bài 4.1** — Trích mã, chữ hoa và độ dài tên khách hàng *(độ khó: dễ)*
- **Bài 4.1** — Tách họ và ghép nhãn tên khách hàng *(độ khó: trung bình)*
- **Bài 4.2** — Số ngày khách hàng đã tham gia *(độ khó: dễ)*
- **Bài 4.2** — Phân loại khách hàng theo năm đăng ký *(độ khó: dễ)*
- **Bài 4.2** — Tính tuổi khách hàng *(độ khó: trung bình)*
- **Bài 4.2** — Khách hàng đăng ký trong 30 ngày qua *(độ khó: trung bình)*
- **Bài 4.2** — Ngày thanh toán T+2 và tháng giao dịch *(độ khó: trung bình)*
- **Bài 4.3** — Phân loại lệnh Mua / Bán / Khác *(độ khó: dễ)*
- **Bài 4.3** — Phân loại khách hàng VIP / Thường theo tài sản *(độ khó: khó)*
- **Bài 4.3** — Tính hoa hồng theo khách hàng mới / cũ *(độ khó: khó)*
- **Bài 4.4** — Lợi suất trong ngày, biên độ và giá trị làm tròn lên tỷ *(độ khó: trung bình)*
- **Bài 4.4** — Độ biến động giá (độ lệch chuẩn) của từng mã *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Gợi ý mở rộng

- Thử `REPLICATE` (T-SQL) / `REPEAT` (MySQL) để lặp ký tự che số điện thoại thay cho `'****'`.
- Tính "tuổi tài khoản" theo tháng bằng `DATEDIFF(month, ...)` / `TIMESTAMPDIFF(MONTH, ...)` và so sánh kết quả với cách tính theo ngày chia 30.
- Tính hoa hồng lũy tiến theo bậc (nhiều nhánh `CASE`): dưới 100 triệu 0,15%, 100–500 triệu 0,1%, trên 500 triệu 0,05%.
- Chuẩn bị Buổi 5 (CTE, bảng tạm, truy vấn con): các bài 4.3 đã dùng truy vấn con tương quan — buổi sau ta sẽ học đầy đủ và tách các bước bằng `WITH ... AS`.

Thực hành trên Windows: kiểu dữ liệu, ngôn ngữ và tiếng Việt

Buổi này có ba chỗ rất hay gặp lỗi khi làm việc trên Windows: **kiểu dữ liệu của cột**, **định dạng ngày** và **mã hóa tiếng Việt**.

1. Xem kiểu dữ liệu của cột bằng giao diện. Trong Object Explorer: `hnx_securities → Tables → dbo.Customers → Columns` (kèm kiểu và độ dài). Hoặc chuột phải bảng → **Design** để xem dạng lưới.

> **Cảnh báo:** chỉ **xem** trong cửa sổ Design, đóng lại bằng nút X và chọn **No** nếu được hỏi lưu. Sửa kiểu dữ liệu bằng tay trên bảng mẫu sẽ làm sai kết quả bài tập.

2. Xem kiểu dữ liệu bằng lệnh.

```sql
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, IS_NULLABLE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Customers'
ORDER BY ORDINAL_POSITION;
```

3. Kiểm tra ngôn ngữ và định dạng ngày của phiên làm việc.

```sql
SELECT @@LANGUAGE AS ngon_ngu;
SELECT CAST('2026-09-25' AS DATE) AS ngay_chuan;        -- an toàn: yyyy-mm-dd với kiểu DATE
SELECT FORMAT(CAST('2026-09-25' AS DATE), 'dd/MM/yyyy') AS ngay_viet_nam;
```

Quy tắc an toàn: viết ngày dạng `'yyyy-mm-dd'` (hoặc `'yyyymmdd'`). Đừng viết `'25/09/2026'` vì cách đọc ngày/tháng phụ thuộc ngôn ngữ đăng nhập (`SET LANGUAGE`, `SET DATEFORMAT`), không phụ thuộc cài đặt vùng của Windows.

4. Tiếng Việt có dấu.

- Cột tiếng Việt dùng `NVARCHAR` (không phải `VARCHAR`); hằng chuỗi có dấu viết kèm tiền tố `N'...'`, ví dụ `N'Hà Nội'`.
- Lưu file `.sql` bằng **UTF-8**: **File → Save As → mũi tên cạnh nút Save → Save with Encoding... → Unicode (UTF-8 with signature)**. Mở bằng Notepad cũ dễ vỡ chữ.
- Kiểm tra nhanh:

```sql
SELECT N'Nguyễn Văn An' AS ten, LEN(N'Nguyễn Văn An') AS do_dai;   -- 13
```

5. Thử các hàm của buổi.

```sql
SELECT full_name,
       UPPER(full_name)                          AS chu_hoa,
       LEN(full_name)                            AS do_dai,
       RIGHT(customer_code, 3)                   AS ba_so_cuoi,
       DATEDIFF(DAY, created_date, '2026-09-25')     AS so_ngay_tham_gia
FROM Customers
ORDER BY customer_id;
```

Lưu ý khóa học dùng ngày cố định `'2026-09-25'` thay cho `GETDATE()` để kết quả luôn giống nhau; khi làm báo cáo thật ở công ty mới thay bằng `GETDATE()`.

Với MySQL Workbench: hàm tương đương là `CHAR_LENGTH`, `DATEDIFF('2026-09-25', created_date)`, `DATE_FORMAT(ngay, '%d/%m/%Y')`; tạo CSDL với `utf8mb4` để lưu tiếng Việt.

Tự kiểm tra: làm **bài 4.2** (số ngày tham gia, tuổi) trên SSMS rồi nộp trên Codelab.

Buổi 5 — CTE, TEMP TABLE và Subquery (truy vấn con)

Về mục lục

Ở các buổi trước, mỗi câu `SELECT` của chúng ta trả lời **một câu hỏi**: "khách hàng nào ở Hà Nội?", "tổng số dư mỗi loại tài khoản là bao nhiêu?". Nhưng bài toán nghiệp vụ trong công ty chứng khoán hiếm khi đơn giản như vậy. Ví dụ phòng Quản lý rủi ro hỏi: *"Những khách hàng nào có tổng tài sản vượt mức trung bình của toàn bộ khách hàng?"* Để trả lời, ta phải **tính trung bình trước**, rồi **mới so sánh** từng khách hàng với con số đó. Một câu hỏi được xây dựng từ kết quả của một câu hỏi khác — đó chính là lý do ta cần **Subquery**, **CTE** và **bảng tạm**.

Bài giảng này dùng lại bộ dữ liệu mẫu của khóa (xem mục "Bộ dữ liệu mẫu của khóa học" ở Buổi 1): các bảng `Customers`, `Accounts`, `Securities`, `MarketPrices`, `Orders`, `Trades`, `Portfolio`, `CustomerStats`, `AuditLog`. "Hôm nay" của bộ dữ liệu cố định là **2026-09-25**. Các ví dụ có nút **Chạy thử** đều chạy trên bộ dữ liệu này — hãy bấm chạy, đọc kết quả rồi mới đọc lời giải thích.

Mục tiêu buổi học

- Sau buổi này bạn có thể phân biệt **subquery vô hướng (scalar)** và **subquery nhiều hàng**, và biết dùng toán tử nào cho từng loại.
- Sau buổi này bạn có thể viết subquery trong `WHERE`, `SELECT`, `FROM` (derived table) và `HAVING`.
- Sau buổi này bạn có thể dùng đúng `IN` / `NOT IN` / `EXISTS` / `NOT EXISTS` / `ALL` / `ANY` / `SOME` và **tránh bẫy `NULL` của `NOT IN`**.
- Sau buổi này bạn có thể phân biệt **correlated** và **non-correlated** subquery và nhận biết chi phí của chúng.
- Sau buổi này bạn có thể chia một bài toán phân tích thành nhiều bước bằng **CTE** (`WITH`, multiple CTE, CTE lồng nhau).
- Sau buổi này bạn có thể dùng **bảng tạm** (`#tmp` / `TEMPORARY TABLE`) cho dữ liệu trung gian và **chọn đúng** giữa Subquery, CTE, TEMP TABLE.

Bản đồ buổi học

Buổi học 180 phút: khoảng 28% lý thuyết + demo (~50 phút) và 72% thực hành (~130 phút, gồm cả thời gian chữa bài). Bài tập nằm **xen kẽ** trong bài giảng — học xong phần nào, làm ngay bài của phần đó. Buổi có 13 bài nên phần lý thuyết giảng gọn, ưu tiên ví dụ chạy trực tiếp; bài khó chưa xong trong giờ thì hoàn thành ở nhà.

Phần	Nội dung	Thời lượng
Mục 1	Subquery là gì, vị trí đặt, scalar vs nhiều hàng	5 phút
Mục 2	Subquery trong `WHERE`: so sánh, `IN` / `NOT IN`, `EXISTS`, `ALL` / `ANY`	13 phút
Mục 3	Subquery trong `SELECT`, `FROM` (derived table), `HAVING`	7 phút
Mục 4	Correlated subquery, chi phí	4 phút
Thực hành 5.1	9 bài về Subquery (5.1a–5.1i)	78 phút
Mục 5	CTE: cú pháp, multiple CTE, CTE lồng nhau	7 phút
Thực hành 5.2	Phân tích nhiều bước bằng CTE (2 bài, có 1 bài khó)	27 phút
Mục 6	TEMP TABLE: tạo, dùng, xóa	5 phút
Thực hành 5.3	Xử lý dữ liệu trung gian bằng bảng tạm	11 phút
Mục 7	So sánh Subquery → CTE → TEMP TABLE trên cùng một bài toán	4 phút
Thực hành 5.4	Bài tổng hợp (bài khó): chọn cách giải tự do	14 phút
Tổng kết	Sai lầm thường gặp, hỏi đáp	5 phút
	Tổng	**180 phút** — lý thuyết + demo ~50 phút, thực hành ~130 phút (72%)

---

1. Subquery là gì? Tại sao cần?

Subquery (truy vấn con) là một câu `SELECT` được đặt **bên trong** một câu truy vấn khác, luôn nằm trong cặp ngoặc đơn `( ... )`. Câu ngoài gọi là **outer query** (truy vấn ngoài).

Ví dụ nhu cầu: tìm tài khoản có số dư lớn hơn mức trung bình. Không có subquery, ta phải làm hai bước thủ công. Bước 1 — tính trung bình:

```
SELECT CAST(ROUND(AVG(balance), 2) AS DECIMAL(18,2)) AS avg_balance FROM Accounts;
```

Kết quả `720290322.58`. Bước 2 — gõ tay con số vừa tính vào truy vấn thứ hai: `SELECT account_id, balance FROM Accounts WHERE balance > 720290322.58;`

Cách này **sai về mặt kỹ thuật** vì con số bị gõ cứng: ngày mai có tài khoản mới, trung bình đổi, báo cáo lập tức sai. Với subquery, hai bước gộp thành một câu và luôn cập nhật:

```sql
SELECT account_id, balance
FROM Accounts
WHERE balance > (SELECT AVG(balance) FROM Accounts)
ORDER BY balance DESC, account_id;
```

account_id	balance
4	8600000000.00
35	8600000000.00
93	8600000000.00
11	3400000000.00
...	... (tổng cộng 24 tài khoản)

> **Ví dụ nghiệp vụ:** phòng Quản lý rủi ro không muốn ai phải "gõ tay" ngưỡng trung bình mỗi sáng. Subquery biến ngưỡng thành **một phần của câu truy vấn**, nên báo cáo luôn đúng với dữ liệu mới nhất.

1.1. Subquery có thể đặt ở đâu?

Vị trí	Vai trò của subquery	Ví dụ ngắn
`WHERE`	Cung cấp giá trị / danh sách để lọc	`WHERE balance > (SELECT AVG(balance) ...)`
`SELECT`	Tính thêm một cột cho mỗi dòng	`SELECT ..., (SELECT COUNT(*) ...) AS num_accounts`
`FROM`	Tạo "bảng tạm thời" trong câu lệnh (derived table)	`FROM (SELECT customer_id, SUM(balance) ... ) t`
`HAVING`	So sánh kết quả nhóm với một giá trị tính riêng	`HAVING SUM(balance) > (SELECT AVG(...) ...)`

1.2. Scalar subquery và subquery nhiều hàng

Phân loại theo **hình dạng kết quả** trả về, vì hình dạng quyết định dùng được toán tử nào:

Loại	Kết quả	Dùng với
Scalar (vô hướng)	1 cột, **đúng 1 dòng** (một giá trị)	`=` , `>` , `<` , `>=` , `<=` , `<>`
Một cột, nhiều dòng	1 cột, 0..n dòng (một danh sách)	`IN` , `NOT IN` , `ANY/SOME` , `ALL`
Nhiều cột, nhiều dòng	một bảng	`EXISTS` , `FROM (...)` (derived table)

> **Cảnh báo:** dùng `=` với subquery trả về **nhiều dòng** là lỗi. SQL Server báo *"Subquery returned more than 1 value"*, MySQL báo *"Subquery returns more than 1 row"*. Khi không chắc chỉ có một dòng, dùng `IN` , hoặc bảo đảm bằng hàm tổng hợp (`MAX` , `AVG` ...) hoặc khóa duy nhất.

> **Lưu ý:** nếu scalar subquery trả về **0 dòng**, kết quả của nó là `NULL` (không phải lỗi). So sánh với `NULL` cho `UNKNOWN`, nên câu truy vấn ngoài **trả về rỗng** — hãy nhớ điều này khi debug "sao báo cáo trống?".

---

2. Subquery trong WHERE

2.1. So sánh với kết quả của subquery vô hướng

Các toán tử `>`, `<`, `=`, `>=`, `<=`, `!=` (hoặc `<>`) đặt trước một subquery trả về **một giá trị**:

Toán tử	Ý nghĩa	Ví dụ
`=`	bằng	`WHERE balance = (SELECT MAX(balance) FROM Accounts)`
`>` / `>=`	lớn hơn / lớn hơn hoặc bằng	`WHERE balance > (SELECT AVG(balance) FROM Accounts)`
`<` / `<=`	nhỏ hơn / nhỏ hơn hoặc bằng	`WHERE close_price <= (SELECT AVG(close_price) FROM MarketPrices)`
`<>` / `!=`	khác	`WHERE customer_id <> (SELECT MIN(customer_id) FROM Customers)`

Ví dụ: số dư lớn nhất trong các tài khoản của khách hàng `customer_id = 6`, và những tài khoản **vượt** mức đó:

```sql
SELECT account_id, customer_id, balance
FROM Accounts
WHERE balance > (SELECT MAX(balance) FROM Accounts WHERE customer_id = 6)
ORDER BY balance DESC, account_id;
```

Subquery `(SELECT MAX(balance) ...)` chỉ trả về **một giá trị** (780.000.000), nên dùng được với `>`.

2.2. IN và NOT IN

`IN` kiểm tra một giá trị có nằm **trong danh sách** mà subquery trả về không. Ví dụ: những mã chứng khoán hiện đang có lệnh chờ khớp:

```sql
SELECT symbol, company_name
FROM Securities
WHERE symbol IN (SELECT symbol FROM Orders WHERE status = 'PENDING')
ORDER BY symbol;
```

Kết quả gồm 9 mã: CEO, FPT, HPG, IDC, MBS, NVB, PVS, VCB, VCS.

`IN` có thể **lồng nhiều tầng**: "khách hàng ← tài khoản ← lệnh":

```sql
SELECT customer_id, full_name
FROM Customers
WHERE customer_id IN (SELECT customer_id FROM Accounts
                      WHERE account_id IN (SELECT account_id FROM Orders WHERE status =
        'MATCHED'))
ORDER BY customer_id;
```

Về kết quả, câu này tương đương một `JOIN` + `DISTINCT`, nhưng đọc lên rất giống câu hỏi nghiệp vụ. Ưu điểm của `IN`: **không bao giờ nhân dòng** (một khách hàng có 10 lệnh vẫn chỉ xuất hiện một lần), trong khi `JOIN` sẽ lặp lại khách hàng đó 10 lần nếu ta quên `DISTINCT`.

Bẫy NOT IN và giá trị NULL

Đây là lỗi **kinh điển** nhất của người mới. Bài toán: tìm khách hàng **chưa giới thiệu ai**, tức `customer_id` không nằm trong cột `referrer_id`. Cách viết trực quan — hãy đếm xem có bao nhiêu khách hàng:

```sql
SELECT COUNT(*) AS so_khach_hang
FROM Customers
WHERE customer_id NOT IN (SELECT referrer_id FROM Customers);
```

Kết quả là **0** (cả SQL Server lẫn MySQL) — trong khi thực tế phải có nhiều khách hàng chưa giới thiệu ai. Vì sao? Cột `referrer_id` chứa nhiều `NULL` (khách hàng không do ai giới thiệu). `x NOT IN (1, 2, NULL)` được đánh giá là `x <> 1 AND x <> 2 AND x <> NULL`; điều kiện cuối luôn là `UNKNOWN`, nên **cả biểu thức không bao giờ là TRUE**.

`x`	`x NOT IN (1, 2)`	`x NOT IN (1, 2, NULL)`
3	TRUE	UNKNOWN (dòng bị loại)
1	FALSE	FALSE

Có hai cách sửa. Cách 1: loại `NULL` khỏi danh sách:

```sql
SELECT COUNT(*) AS so_khach_hang
FROM Customers
WHERE customer_id NOT IN (SELECT referrer_id FROM Customers WHERE referrer_id IS NOT NULL);
```

Cách 2 (khuyến nghị): `NOT EXISTS` — không nhạy cảm với `NULL`:

```sql
SELECT c.customer_id, c.full_name
FROM Customers c
WHERE NOT EXISTS (SELECT 1 FROM Customers r WHERE r.referrer_id = c.customer_id)
ORDER BY c.customer_id;
```

Cả hai cho 35 khách hàng (40 khách hàng trừ 5 người đã giới thiệu: ID 1, 2, 3, 8, 31).

> **Mẹo:** khi viết `NOT IN (subquery)`, luôn tự hỏi *"cột trong subquery có thể NULL không?"*. Nếu có → dùng `NOT EXISTS`. Đây là quy tắc vàng cần thuộc lòng.

Vừa học xong subquery vô hướng, `IN` (kể cả lồng nhau) và bẫy `NOT IN` với `NULL`, hãy làm ngay bài **5.1a** (tài khoản trên mức trung bình), **5.1b** (`IN` lồng nhau) và **5.1d** (khách hàng chưa giới thiệu ai — tránh bẫy `NULL`).

- **Bài 5.1** — Tài khoản có số dư trên mức trung bình *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng có lệnh đã khớp (IN lồng nhau) *(độ khó: dễ)*
- **Bài 5.1** — Chứng khoán chưa có lệnh chờ khớp (NOT EXISTS) *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng chưa giới thiệu ai (bẫy NOT IN với NULL) *(độ khó: trung bình)*
- **Bài 5.1** — Số dư vượt mọi tài khoản của một khách hàng (ALL) *(độ khó: trung bình)*
- **Bài 5.1** — Hồ sơ khách hàng với cột tính từ subquery (SELECT) *(độ khó: trung bình)*
- **Bài 5.1** — Tài sản trung bình theo thành phố (Derived Table) *(độ khó: trung bình)*
- **Bài 5.1** — Khách hàng có tổng tài sản trên trung bình (HAVING) *(độ khó: trung bình)*
- **Bài 5.1** — Tài khoản vượt trung bình cùng loại (Correlated) *(độ khó: trung bình)*

2.3. EXISTS và NOT EXISTS

`EXISTS (subquery)` trả về TRUE nếu subquery **có ít nhất một dòng**, bất kể nội dung dòng đó. Vì vậy người ta thường viết `SELECT 1` bên trong. Subquery của `EXISTS` thường **tham chiếu tới bảng ngoài** (correlated, xem mục 4):

```sql
-- Khách hàng có ít nhất một lệnh khớp một phần (PARTIAL)
SELECT c.customer_id, c.full_name
FROM Customers c
WHERE EXISTS (SELECT 1
              FROM Accounts a JOIN Orders o ON o.account_id = a.account_id
              WHERE a.customer_id = c.customer_id AND o.status = 'PARTIAL')
ORDER BY c.customer_id;
```

Kết quả gồm các khách hàng 2, 5, 6, 16, 18. Và `NOT EXISTS` cho phép tìm "những thứ chưa từng có":

```sql
-- Khách hàng chưa mở tài khoản nào
SELECT c.customer_id, c.full_name
FROM Customers c
WHERE NOT EXISTS (SELECT 1 FROM Accounts a WHERE a.customer_id = c.customer_id)
ORDER BY c.customer_id;
```

customer_id	full_name
10	Ngo Thi Nga
15	Vo Van Tuan
28	Dang Thi Linh

`NOT EXISTS` cũng là mẫu của bài **5.1c** (mã chứng khoán chưa có lệnh `PENDING`). So sánh hai toán tử:

	`IN`	`EXISTS`
Ý nghĩa	giá trị có thuộc danh sách?	có tồn tại dòng thỏa điều kiện?
Subquery trả về	1 cột	bất kỳ (thường `SELECT 1`)
Xử lý NULL	`NOT IN` bị "nhiễm" NULL	`NOT EXISTS` an toàn
Dùng khi	danh sách giá trị ngắn, đơn giản	điều kiện phức tạp, nhiều cột, cần tương quan

2.4. ALL, ANY, SOME

Các từ khóa này đặt **sau** toán tử so sánh để so sánh với **cả một danh sách**:

- `> ALL (danh sách)`: lớn hơn **mọi** phần tử (tức lớn hơn giá trị lớn nhất).
- `> ANY (danh sách)` hay `> SOME (...)`: lớn hơn **ít nhất một** phần tử (tức lớn hơn giá trị nhỏ nhất). `SOME` là tên khác của `ANY`.
- `= ANY (...)` tương đương `IN (...)`.
- `<> ALL (...)` tương đương `NOT IN (...)`.

Ví dụ: tài khoản có số dư lớn hơn **tất cả** tài khoản của khách hàng 6 (số dư 780 triệu, 410 triệu, 410 triệu):

```sql
SELECT account_id, balance
FROM Accounts
WHERE balance > ALL (SELECT balance FROM Accounts WHERE customer_id = 6)
ORDER BY balance DESC, account_id;
```

Kết quả gồm 16 tài khoản, bắt đầu bằng tài khoản 4, 35, 93 (8,6 tỷ). Ví dụ với `ANY`: giá đóng cửa 2026-09-25 của mã nào **cao hơn giá vốn của ít nhất một mã** mà tài khoản 1 đang nắm (giá vốn 25.500, 24.700, 52.000, 60.700)?

```sql
SELECT symbol, close_price
FROM MarketPrices
WHERE trade_date = '2026-09-25'
  AND close_price > ANY (SELECT avg_cost FROM Portfolio WHERE account_id = 1)
ORDER BY symbol;
```

Chỉ những mã có giá đóng cửa cao hơn 24.700 (giá vốn nhỏ nhất) được chọn: 9 mã gồm FPT, HPG, IDC, MBS, PVI, PVS, VCB, VCS, VNM.

> **Lưu ý:** `> ALL (danh sách rỗng)` là **TRUE** (vì "không có phần tử nào vi phạm"), còn `> ANY (danh sách rỗng)` là **FALSE**. Đây là chỗ hay gây bất ngờ. Nếu danh sách chứa NULL, `ALL` cũng có thể cho kết quả UNKNOWN giống `NOT IN`.

Vừa học xong `EXISTS` / `NOT EXISTS` và `ALL` / `ANY`, hãy làm bài **5.1c** (mã chứng khoán chưa có lệnh chờ khớp, dùng `NOT EXISTS`) và **5.1e** (số dư vượt mọi tài khoản của một khách hàng, dùng `ALL`).

- **Bài 5.1** — Tài khoản có số dư trên mức trung bình *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng có lệnh đã khớp (IN lồng nhau) *(độ khó: dễ)*

- **Bài 5.1** — Chứng khoán chưa có lệnh chờ khớp (NOT EXISTS) *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng chưa giới thiệu ai (bẫy NOT IN với NULL) *(độ khó: trung bình)*
- **Bài 5.1** — Số dư vượt mọi tài khoản của một khách hàng (ALL) *(độ khó: trung bình)*
- **Bài 5.1** — Hồ sơ khách hàng với cột tính từ subquery (SELECT) *(độ khó: trung bình)*
- **Bài 5.1** — Tài sản trung bình theo thành phố (Derived Table) *(độ khó: trung bình)*
- **Bài 5.1** — Khách hàng có tổng tài sản trên trung bình (HAVING) *(độ khó: trung bình)*
- **Bài 5.1** — Tài khoản vượt trung bình cùng loại (Correlated) *(độ khó: trung bình)*

---

3. Subquery trong SELECT và FROM

3.1. Cột tính toán từ subquery (trong SELECT)

Subquery vô hướng có thể đặt trong danh sách cột để tạo **cột thống kê đi kèm mỗi dòng**:

```
SELECT c.customer_id, c.full_name,
       (SELECT COUNT(*) FROM Accounts a WHERE a.customer_id = c.customer_id) AS
        num_accounts,
       COALESCE((SELECT SUM(a.balance) FROM Accounts a WHERE a.customer_id = c.customer_id),
        0) AS total_balance
FROM Customers c
WHERE c.customer_id <= 5
ORDER BY c.customer_id;
```

customer_id	full_name	num_accounts	total_balance
1	Nguyen Van An	1	67000000.00
2	Tran Thi Binh	3	8640000000.00
3	Le Van Cuong	3	168000000.00
4	Pham Thi Dung	2	1480000000.00
5	Hoang Van Em	2	3432000000.00

Ưu điểm: giữ nguyên **mỗi khách hàng một dòng**, kể cả khách hàng chưa có tài khoản (với `INNER JOIN` họ sẽ biến mất; với `LEFT JOIN` + `GROUP BY` thì phức tạp hơn). Chú ý `COALESCE(..., 0)` vì `SUM` trên tập rỗng trả `NULL`. Subquery trong `SELECT` **bắt buộc là scalar**: nếu trả về nhiều dòng sẽ báo lỗi.

3.2. Derived table (subquery trong FROM)

Subquery đặt ở `FROM` biến thành một **bảng ảo** tồn tại trong suốt câu lệnh, và **bắt buộc phải có alias** (cả SQL Server lẫn MySQL). Rất hữu ích khi cần **gom nhóm hai lần** (tổng theo khách hàng, rồi trung bình theo thành phố):

```sql
SELECT c.city, COUNT(*) AS num_customers,
       CAST(ROUND(AVG(t.total_assets), 2) AS DECIMAL(18,2)) AS avg_assets
FROM Customers c
JOIN (SELECT customer_id, SUM(balance) AS total_assets
      FROM Accounts GROUP BY customer_id) t
  ON t.customer_id = c.customer_id
GROUP BY c.city
ORDER BY avg_assets DESC, c.city;
```

city	num_customers	avg_assets
Hai Phong	3	4601333333.33
TP.HCM	7	3167000000.00
Da Nang	4	2672000000.00
Can Tho	3	901666666.67
Ha Noi	20	881050000.00

Không thể viết `AVG(SUM(balance))` trực tiếp — hàm tổng hợp lồng nhau không hợp lệ — nên derived table là lời giải tự nhiên. Cách đọc: bên trong tính **tổng tài sản mỗi khách hàng** (bảng ảo `t`), bên ngoài nối với `Customers` để lấy thành phố rồi tính trung bình theo thành phố.

Alias cho derived table là bắt buộc: thiếu alias, SQL Server báo *"Incorrect syntax near ')'"*, MySQL báo *"Every derived table must have its own alias"*. Mọi cột của derived table cũng phải có **tên** (dùng `AS`).

> **SQL Server vs MySQL:** trong derived table, SQL Server **không cho** `ORDER BY` nếu không kèm `TOP` / `OFFSET` (báo lỗi *"The ORDER BY clause is invalid in views, inline functions, derived tables, subqueries..."*); MySQL cho phép nhưng bỏ qua. Vì vậy hãy đặt `ORDER BY` ở truy vấn **ngoài cùng**.

Vừa học xong subquery trong `SELECT` (cột tính từ subquery) và trong `FROM` (derived table), hãy làm bài **5.1f** (hồ sơ khách hàng với cột tính từ subquery) và **5.1g** (tài sản trung bình theo thành phố).

- **Bài 5.1** — Tài khoản có số dư trên mức trung bình *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng có lệnh đã khớp (IN lồng nhau) *(độ khó: dễ)*
- **Bài 5.1** — Chứng khoán chưa có lệnh chờ khớp (NOT EXISTS) *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng chưa giới thiệu ai (bẫy NOT IN với NULL) *(độ khó: trung bình)*
- **Bài 5.1** — Số dư vượt mọi tài khoản của một khách hàng (ALL) *(độ khó: trung bình)*
- **Bài 5.1** — Hồ sơ khách hàng với cột tính từ subquery (SELECT) *(độ khó: trung bình)*
- **Bài 5.1** — Tài sản trung bình theo thành phố (Derived Table) *(độ khó: trung bình)*
- **Bài 5.1** — Khách hàng có tổng tài sản trên trung bình (HAVING) *(độ khó: trung bình)*
- **Bài 5.1** — Tài khoản vượt trung bình cùng loại (Correlated) *(độ khó: trung bình)*

3.3. Subquery trong HAVING

`HAVING` lọc **nhóm** sau `GROUP BY`; ta có thể so sánh tổng của nhóm với một con số tính từ subquery:

```sql
SELECT c.customer_id, c.full_name, SUM(a.balance) AS total_assets
FROM Customers c JOIN Accounts a ON a.customer_id = c.customer_id
GROUP BY c.customer_id, c.full_name
HAVING SUM(a.balance) > (SELECT AVG(t.total)
                         FROM (SELECT SUM(balance) AS total FROM Accounts GROUP BY
        customer_id) t)
ORDER BY total_assets DESC, c.customer_id;
```

Trung bình tổng tài sản/khách hàng khoảng 1,81 tỷ, nên 7 khách hàng lọt vào: 17 (12,067 tỷ), 40 (9,5 tỷ), 2 (8,64 tỷ), 34, 38, 32, 5.

> **Lưu ý:** cần phân biệt trung bình **số dư tài khoản** (khoảng 0,72 tỷ) với trung bình **tổng tài sản khách hàng** (1,81 tỷ). Bài **5.1h** kiểm tra bạn có hiểu đúng "mức trung bình của nhóm" hay không.

3.4. Khi nào dùng subquery ở SELECT và FROM?

Vị trí	Dùng khi	Lưu ý
`SELECT`	cần 1–3 con số thống kê phụ và muốn giữ nguyên số dòng của bảng chính	phải là scalar; mỗi cột là một lần quét bảng phụ — quá nhiều cột thì chuyển sang `JOIN`
`FROM`	cần **tính trước một bước** (gom nhóm, xếp hạng, lọc) rồi mới nối/gom tiếp	bắt buộc alias; đặt `ORDER BY` ở ngoài cùng
`HAVING`	so sánh kết quả nhóm với một con số tính riêng	phân biệt "trung bình theo dòng" và "trung bình theo nhóm"

---

4. Correlated Subquery (subquery tương quan)

Non-correlated subquery độc lập với truy vấn ngoài: chạy được riêng lẻ (như `SELECT AVG(balance) FROM Accounts`). Về nguyên tắc chỉ cần tính **một lần**.

Correlated subquery tham chiếu tới cột của truy vấn ngoài, nên **không chạy riêng được** và về mặt logic phải tính **lại cho từng dòng** của truy vấn ngoài.

	Non-correlated	Correlated
Chạy riêng được?	có	không (dùng cột của bảng ngoài)
Số lần tính (về logic)	1 lần	1 lần cho mỗi dòng ngoài
Ví dụ	`WHERE balance > (SELECT AVG(balance) FROM Accounts)`	`WHERE balance > (SELECT AVG(b.balance) FROM Accounts b WHERE b.account_type = a.account_type)`

Bài toán kinh điển: tìm tài khoản có số dư lớn hơn **mức trung bình của cùng loại** (`CASH` / `MARGIN`):

```sql
SELECT a.account_id, a.account_type, a.balance,
       CAST(ROUND((SELECT AVG(b.balance) FROM Accounts b
                   WHERE b.account_type = a.account_type), 2) AS DECIMAL(18,2)) AS type_avg
FROM Accounts a
WHERE a.balance > (SELECT AVG(b.balance) FROM Accounts b WHERE b.account_type =
        a.account_type)
ORDER BY a.account_type, a.balance DESC, a.account_id;
```

Trung bình của `CASH` là 831.875.000, của `MARGIN` là 551.405.405,41; mỗi tài khoản được so với mức của đúng loại của nó. Chú ý cách đặt alias khác nhau (`a` cho bảng ngoài, `b` cho bảng trong) — nếu quên alias, điều kiện `account_type = account_type` luôn đúng và bạn so sánh với trung bình toàn bảng.

Chi phí và lưu ý

- Về logic, subquery tương quan chạy N lần (N = số dòng ngoài). Trình tối ưu của SQL Server và MySQL 8 thường tự biến đổi thành join/semi-join, nhưng không phải lúc nào cũng vậy.
- Với bảng lớn (hàng triệu lệnh), hãy đặt **index** trên cột tương quan (`Accounts(account_type)` , `Orders(account_id)`).
- Nếu thấy chậm, thử viết lại thành `JOIN` với derived table (tính trung bình một lần cho mỗi loại, nối lại) hoặc dùng CTE (mục 5).
- `EXISTS` / `NOT EXISTS` dừng ngay khi tìm thấy dòng đầu tiên nên thường rất hiệu quả.

> **Mẹo:** cách nhận biết nhanh một subquery là correlated: xóa nó khỏi câu lệnh, dán riêng vào cửa sổ Query. Nếu báo lỗi *"invalid column"* (thiếu cột của bảng ngoài) thì nó là correlated.

Vừa học xong subquery trong `HAVING` (mục 3.3) và correlated subquery, hãy làm bài **5.1h** (khách hàng có tổng tài sản trên trung bình) và **5.1i** (tài khoản vượt trung bình cùng loại) để kết thúc phần Subquery trước khi sang CTE.

- **Bài 5.1** — Tài khoản có số dư trên mức trung bình *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng có lệnh đã khớp (IN lồng nhau) *(độ khó: dễ)*
- **Bài 5.1** — Chứng khoán chưa có lệnh chờ khớp (NOT EXISTS) *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng chưa giới thiệu ai (bẫy NOT IN với NULL) *(độ khó: trung bình)*
- **Bài 5.1** — Số dư vượt mọi tài khoản của một khách hàng (ALL) *(độ khó: trung bình)*
- **Bài 5.1** — Hồ sơ khách hàng với cột tính từ subquery (SELECT) *(độ khó: trung bình)*
- **Bài 5.1** — Tài sản trung bình theo thành phố (Derived Table) *(độ khó: trung bình)*
- **Bài 5.1** — Khách hàng có tổng tài sản trên trung bình (HAVING) *(độ khó: trung bình)*
- **Bài 5.1** — Tài khoản vượt trung bình cùng loại (Correlated) *(độ khó: trung bình)*

---

5. CTE — Common Table Expression

5.1. Khái niệm và cú pháp WITH

CTE là một "bảng tạm có tên" chỉ tồn tại trong **một câu lệnh**, khai báo bằng `WITH` :

```sql
WITH ten_cte AS (
    SELECT ...
)
SELECT ... FROM ten_cte ...;
```

Ví dụ: khách hàng có tổng tài sản trên 5 tỷ:

```sql
WITH t AS (
    SELECT customer_id, SUM(balance) AS tot
    FROM Accounts
    GROUP BY customer_id
)
SELECT customer_id, tot FROM t WHERE tot > 5000000000 ORDER BY tot DESC;
```

customer_id	tot
17	12067000000.00
40	9500000000.00
2	8640000000.00

> **SQL Server vs MySQL:** với SQL Server, nếu `WITH` không phải câu lệnh đầu tiên trong batch, câu **trước nó phải kết thúc bằng dấu `;`**, nếu không sẽ báo *"Incorrect syntax near the keyword 'WITH'"*. Thói quen tốt: luôn kết thúc câu lệnh bằng `;`. MySQL hỗ trợ CTE từ 8.0 (bộ Judge0 của khóa dùng MySQL 8).

Trước khi so sánh CTE với subquery, hãy nhìn một CTE nhiều bước như một quy trình: bước sau đọc kết quả của bước trước.

Hình minh họa: Luồng chạy của CTE (xem hình động trên trang bài giảng trực tuyến).

5.2. CTE so với Subquery

Tiêu chí	Subquery / Derived table	CTE
Đọc từ trên xuống	khó nếu lồng nhiều tầng (đọc từ trong ra ngoài)	dễ: từng bước có **tên**
Dùng lại trong cùng câu	phải viết lại từng lần	gọi tên nhiều lần
Đệ quy	không	có (recursive CTE)
Lưu kết quả	không	**không** (chỉ là cách viết; có thể bị tính lại mỗi lần được nhắc tới)

5.3. Multiple CTE và CTE lồng nhau

Nhiều CTE ngăn cách bởi dấu phẩy, **chỉ một từ khóa** `WITH`. CTE sau được phép tham chiếu CTE trước (đây là dạng "lồng nhau" thường gặp). Báo cáo phân tầng khách hàng kiểu bài **5.2a** là ví dụ điển hình — chia thành 4 bước: tổng tài sản → trung bình → phân loại → lọc nhóm mục tiêu:

```
WITH total_assets AS (                      -- Bước 1: tổng tài sản từng khách hàng (tài
        khoản ACTIVE)
    SELECT customer_id, SUM(balance) AS total
    FROM Accounts WHERE status = 'ACTIVE' GROUP BY customer_id
), avg_assets AS (                          -- Bước 2: trung bình (dùng CTE ở bước 1)
    SELECT AVG(total) AS avg_total FROM total_assets
), tiers AS (                               -- Bước 3: phân loại
    SELECT t.customer_id, t.total, a.avg_total,
           CASE WHEN t.total >= 2 * a.avg_total THEN 'CAO'
                WHEN t.total >= a.avg_total     THEN 'TRUNG_BINH'
                ELSE 'THAP' END AS tier
    FROM total_assets t CROSS JOIN avg_assets a
)
SELECT customer_id, total AS total_assets,   -- Bước 4: lọc nhóm mục tiêu
       CAST(ROUND(avg_total, 2) AS DECIMAL(18,2)) AS avg_assets, tier
FROM tiers WHERE tier = 'CAO'
ORDER BY total DESC, customer_id;
```

Kết quả: 6 khách hàng nhóm CAO (17, 40, 2, 34, 38, 32) so với mức trung bình 1.826.628.571,43.

> **Mẹo:** mỗi bước có thể kiểm tra riêng bằng cách đổi câu cuối thành `SELECT * FROM <tên CTE>` — kỹ thuật debug rất hữu ích. Thử: thay câu `SELECT` cuối bằng `SELECT * FROM tiers ORDER BY customer_id;` để xem toàn bộ khách hàng và tầng của họ.

Bài **5.2b** nâng cấp lên bài toán "giá trị danh mục theo giá thị trường" với chuỗi CTE `px → acc_value → cust_value → summary`, trong đó mỗi CTE dựa vào CTE trước.

5.4. Tái sử dụng kết quả trong một câu SQL

Một CTE có thể được dùng **nhiều lần** trong câu lệnh cuối; ví dụ `total_assets` ở trên vừa dùng để tính trung bình vừa để phân loại. Với subquery ta phải sao chép đoạn code hai lần:

```
WITH t AS (SELECT customer_id, SUM(balance) AS total FROM Accounts GROUP BY customer_id)
SELECT customer_id, total AS total_assets
FROM t
WHERE total > (SELECT AVG(total) FROM t)   -- dùng lại t lần thứ hai
ORDER BY total DESC, customer_id;
```

> **Lưu ý:** CTE **không** bảo đảm được "cache" — trình tối ưu có thể tính lại mỗi lần nhắc đến. Nếu bước tính tốn kém và dùng nhiều lần, hãy cân nhắc bảng tạm (mục 6).

5.5. Mở rộng: CTE đệ quy (giới thiệu)

CTE có thể **tự tham chiếu** để duyệt cấu trúc cây. Ví dụ: chuỗi người được giới thiệu (trực tiếp và gián tiếp) từ khách hàng 1. Cú pháp khác nhau giữa hai hệ:

SQL Server — chỉ cần `WITH` (không có từ `RECURSIVE`):

```
WITH chain AS (
    SELECT customer_id, referrer_id, 1 AS lvl FROM Customers WHERE referrer_id = 1
    UNION ALL
    SELECT c.customer_id, c.referrer_id, ch.lvl + 1
    FROM Customers c JOIN chain ch ON c.referrer_id = ch.customer_id
)
SELECT customer_id, lvl FROM chain ORDER BY lvl, customer_id;
```

MySQL — thêm từ khóa `RECURSIVE`:

```
WITH RECURSIVE chain AS (
    SELECT customer_id, referrer_id, 1 AS lvl FROM Customers WHERE referrer_id = 1
    UNION ALL
    SELECT c.customer_id, c.referrer_id, ch.lvl + 1
    FROM Customers c JOIN chain ch ON c.referrer_id = ch.customer_id
)
SELECT customer_id, lvl FROM chain ORDER BY lvl, customer_id;
```

Kết quả: khách hàng 8 và 11 (cấp 1), 27 và 29 (cấp 2). Đây là phần mở rộng, không nằm trong bài tập chấm điểm.

Bạn đã nắm CTE: cú pháp `WITH`, multiple CTE, CTE lồng nhau và tái sử dụng. Vừa học xong phần này, hãy làm các bài **5.2a** (báo cáo phân tầng khách hàng, 4 bước) và **5.2b** (giá trị danh mục theo giá thị trường, chuỗi CTE) để tập chia một bài toán lớn thành nhiều bước có tên.

- **Bài 5.2** — Báo cáo phân tầng khách hàng bằng CTE nhiều bước *(độ khó: trung bình)*
- **Bài 5.2** — Giá trị danh mục theo giá thị trường (multiple/nested CTE) *(độ khó: khó)*

---

6. TEMP TABLE — Bảng tạm

6.1. Khái niệm và mục đích

Bảng tạm là bảng thật (có cấu trúc, có thể có index) nhưng **chỉ tồn tại trong phiên làm việc (session)** của bạn và tự biến mất khi đóng phiên. Khác với CTE (chỉ sống trong một câu lệnh), bảng tạm **sống qua nhiều câu lệnh**, và kết quả được **lưu thật** nên không bị tính lại. Dùng khi:

- kết quả trung gian **tốn nhiều thời gian** để tính và được dùng nhiều lần;
- cần **nhiều bước** xử lý, mỗi bước kiểm tra riêng;
- cần thêm index vào tập trung gian, hoặc cập nhật (`UPDATE`) dữ liệu trung gian.

> **Ví dụ nghiệp vụ:** chốt danh sách khách hàng đủ điều kiện ưu đãi phí vào một bảng tạm, sau đó dùng danh sách đó cho ba báo cáo khác nhau — không phải tính lại điều kiện phức tạp mỗi lần.

6.2. Tạo, dùng và xóa

SQL Server — tên bắt đầu bằng `#` (riêng theo phiên) hoặc `##` (dùng chung toàn máy chủ, hiếm dùng):

```
-- Cách 1: tạo và đổ dữ liệu cùng lúc (SELECT ... INTO)
SELECT c.customer_id, SUM(a.balance) AS total_assets
INTO #eligible
FROM Customers c JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.status = 'ACTIVE' AND a.status = 'ACTIVE'
GROUP BY c.customer_id
HAVING SUM(a.balance) >= 100000000;

-- Dùng bảng tạm: ba khách hàng đứng đầu
SELECT TOP 3 customer_id, total_assets FROM #eligible ORDER BY total_assets DESC,
        customer_id;
```

Cách 2: khai báo cấu trúc rồi `INSERT ... SELECT` (kiểm soát kiểu dữ liệu, thêm khóa chính), và xóa sớm khi xong (không bắt buộc):

```
CREATE TABLE #eligible2 (customer_id INT PRIMARY KEY, total_assets DECIMAL(18,2));
INSERT INTO #eligible2 SELECT customer_id, SUM(balance) FROM Accounts GROUP BY customer_id;
DROP TABLE IF EXISTS #eligible;
```

MySQL — thêm từ khóa `TEMPORARY`, không dùng ký hiệu `#`:

```
CREATE TEMPORARY TABLE eligible AS
SELECT c.customer_id, SUM(a.balance) AS total_assets
FROM Customers c JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.status = 'ACTIVE' AND a.status = 'ACTIVE'
GROUP BY c.customer_id
HAVING SUM(a.balance) >= 100000000;

SELECT customer_id, total_assets FROM eligible ORDER BY total_assets DESC, customer_id LIMIT
        3;
```

Xóa bảng tạm trong MySQL: `DROP TEMPORARY TABLE IF EXISTS eligible;`. Với dữ liệu mẫu, bảng `eligible` gồm 25 khách hàng; khách hàng 40 (9,5 tỷ), 2 (8,64 tỷ), 34 (4,69 tỷ) là ba người đứng đầu.

Khác biệt cần nhớ

	SQL Server	MySQL
Ký hiệu	`#tên` (phiên), `##tên` (toàn cục)	`CREATE TEMPORARY TABLE tên`
Tạo từ SELECT	`SELECT ... INTO #t`	`CREATE TEMPORARY TABLE t AS SELECT ...`
Xóa	`DROP TABLE #t`	`DROP TEMPORARY TABLE t`
Nhắc tới nhiều lần trong **một** truy vấn	được	**không** (lỗi *"Can't reopen table"*)
Tên trùng bảng thật	tên `#` không trùng	bảng tạm **che khuất** bảng thật cùng tên

> **Cảnh báo (MySQL):** lỗi "Can't reopen table" xảy ra khi bạn `JOIN` một bảng tạm với chính nó hoặc dùng nó trong cả truy vấn ngoài lẫn subquery. Cách khắc phục: tạo bảng tạm thứ hai (như bài **5.3a** làm) hoặc dùng CTE.

Ngoài `#tmp`, SQL Server còn có **biến bảng** `DECLARE @t TABLE (...)` (nhẹ hơn, phạm vi trong batch) — tham khảo thêm khi làm stored procedure ở các buổi sau.

6.3. TEMP TABLE, CTE và Derived table — chọn cái nào?

Tiêu chí	Derived table	CTE	Bảng tạm
Phạm vi sống	trong một câu	trong một câu	cả phiên
Dùng lại nhiều câu lệnh	không	không	**có**
Lưu kết quả thật	không	không	**có**
Đánh index	không	không	**có**
Độ dễ đọc	thấp nếu lồng sâu	**cao**	cao (từng bước rõ ràng)
Cần quyền tạo bảng	không	không	tạo trong tempdb (SQL Server)
Phù hợp	bước gom nhóm đơn giản	phân tích nhiều bước trong **một** báo cáo	dữ liệu trung gian lớn, dùng lại, debug từng bước

Quy tắc chọn nhanh: bắt đầu bằng subquery/CTE (đơn giản, không "rác" dữ liệu). Chuyển sang bảng tạm khi kết quả trung gian **lớn, tốn kém, và được dùng lại** trong nhiều câu lệnh.

Vừa học xong bảng tạm, hãy làm bài **5.3a**: tạo bảng tạm chứa tập khách hàng đủ điều kiện, rồi dùng nó cho bước phân tích tiếp theo.

- **Bài 5.3** — Phân tích nhiều bước bằng bảng tạm *(độ khó: trung bình)*

---

7. Thực hành tổng hợp: Subquery → CTE → TEMP TABLE

Cùng một bài toán, ba cách viết. Bài toán: *"Khách hàng có tổng tài sản lớn hơn hẳn mức trung bình của tất cả khách hàng"* (kết quả gồm 7 khách hàng, đứng đầu là 17, 40, 2).

Cách 1 — Subquery: gọn nhất khi logic đơn giản (mục 3.3).

```
SELECT customer_id, SUM(balance) AS total_assets
FROM Accounts
GROUP BY customer_id
HAVING SUM(balance) > (SELECT AVG(t.total) FROM (SELECT SUM(balance) AS total FROM Accounts
        GROUP BY customer_id) t)
ORDER BY total_assets DESC, customer_id;
```

Cách 2 — CTE: dễ đọc, tổng tài sản chỉ khai báo **một lần**.

```
WITH t AS (SELECT customer_id, SUM(balance) AS total FROM Accounts GROUP BY customer_id)
SELECT customer_id, total AS total_assets
FROM t
WHERE total > (SELECT AVG(total) FROM t)
ORDER BY total DESC, customer_id;
```

Cách 3 — TEMP TABLE: từng bước tách rời, kiểm tra được. SQL Server:

```
SELECT customer_id, SUM(balance) AS total INTO #t FROM Accounts GROUP BY customer_id;

SELECT customer_id, total AS total_assets FROM #t
WHERE total > (SELECT AVG(total) FROM #t)
ORDER BY total DESC, customer_id;
```

MySQL **không** dùng được một bảng tạm hai lần trong một truy vấn, nên phải tính trung bình sang bảng tạm thứ hai:

```
CREATE TEMPORARY TABLE t  AS SELECT customer_id, SUM(balance) AS total FROM Accounts GROUP
        BY customer_id;
CREATE TEMPORARY TABLE av AS SELECT AVG(total) AS avg_total FROM t;

SELECT t.customer_id, t.total AS total_assets FROM t JOIN av ON t.total > av.avg_total
ORDER BY t.total DESC, t.customer_id;
```

Cả ba cho **cùng kết quả**. So sánh:

Cách	Dễ đọc	Tái sử dụng	Hiệu năng (dữ liệu lớn)	Khi nào chọn
Subquery	vừa (lồng 2 tầng)	không	tốt cho truy vấn đơn	điều kiện lọc đơn giản, một lần
CTE	tốt	trong 1 câu	tương đương subquery	báo cáo nhiều bước trong **một** câu
TEMP TABLE	tốt	nhiều câu	tốt nhất khi dùng lại nhiều lần / có index	dữ liệu trung gian lớn, dùng lại, cần debug từng bước

> **Mẹo:** ba tiêu chí chọn cách viết: (1) **dễ đọc** — CTE thắng khi có từ 3 bước; (2) **tái sử dụng** — nếu kết quả dùng qua nhiều câu lệnh, bảng tạm; (3) **hiệu năng** — đo bằng kế hoạch thực thi trước khi kết luận, đừng đoán.

Bài **5.4a** (khách hàng nắm giữ lớn nhất từng mã) là bài tổng hợp: bạn tự chọn cách giải. Gợi ý: CTE `holdings` (số lượng nắm giữ theo cặp khách hàng–mã), CTE `totals` (lớn nhất và tổng theo mã), rồi nối lại; chú ý xử lý đồng hạng bằng `MIN(customer_id)`.

- **Bài 5.4** — Tổng hợp: khách hàng nắm giữ lớn nhất từng mã *(độ khó: khó)*

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
`NOT IN` với cột có `NULL`	kết quả rỗng oan	`NOT EXISTS`, hoặc thêm `IS NOT NULL` trong subquery
Dùng `=` với subquery trả về nhiều dòng	lỗi *"returned more than 1 value/row"*	dùng `IN`, hoặc `MAX` / `AVG`, hoặc khóa duy nhất
Quên alias cho derived table	lỗi cú pháp	luôn đặt alias: `FROM (...) t`
Quên `;` trước `WITH` trong SQL Server	*Incorrect syntax near the keyword 'WITH'*	kết thúc mọi câu bằng `;`
Correlated subquery không đặt alias	so sánh cột với chính nó	`a` cho bảng ngoài, `b` cho bảng trong
Nhầm "trung bình số dư tài khoản" với "trung bình tổng tài sản khách hàng"	sai nhóm mục tiêu	gom theo khách hàng trước, rồi mới lấy trung bình
MySQL: dùng một bảng tạm hai lần trong một truy vấn	lỗi *Can't reopen table*	tạo bảng tạm thứ hai, hoặc dùng CTE
Dùng `GETDATE()` / `NOW()` để lấy "hôm nay"	kết quả đổi theo ngày, chấm sai	dùng ngày cố định `'2026-09-25'`
Chia số nguyên: `100 * a / b` trong T-SQL	kết quả nguyên, mất phần thập phân	`100.0 * a / b` và `CAST(ROUND(x, 2) AS DECIMAL(18,2))`

Tóm tắt

Chủ đề	Ghi nhớ
Scalar subquery	1 giá trị; dùng với `= > < ...`
`IN` / `NOT IN`	danh sách 1 cột; `NOT IN` + `NULL` = bẫy
`EXISTS` / `NOT EXISTS`	có/không tồn tại dòng; an toàn với `NULL`
`ALL` / `ANY` / `SOME`	so sánh với cả danh sách; `> ALL` = lớn hơn MAX
SELECT subquery	cột thống kê phụ; phải scalar
Derived table	subquery ở `FROM`, bắt buộc alias
Correlated	tham chiếu bảng ngoài; nhớ alias và index
CTE	`WITH a AS (...), b AS (...)` ; đọc từng bước; sống trong 1 câu
TEMP TABLE	`#t` / `TEMPORARY TABLE` ; sống cả phiên, lưu thật

Gợi ý mở rộng

- Viết lại bài 5.1i bằng `JOIN` với derived table và so sánh thời gian chạy (bật `SET STATISTICS TIME ON` trong SSMS).
- Thử bài 5.2a với các ngưỡng phân loại khác (1,5 lần, 3 lần trung bình) và thêm nhóm `'TRUNG_BINH'` vào kết quả.
- Dùng CTE đệ quy để tính "cây giới thiệu" đầy đủ của chương trình *Bạn giới thiệu bạn*.
- Tìm hiểu hàm cửa sổ (`ROW_NUMBER` , `RANK`) — cách hiện đại để giải "top-N mỗi nhóm" mà không cần subquery tương quan.

Bài tập thực hành của buổi

- **Bài 5.1** — Tài khoản có số dư trên mức trung bình *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng có lệnh đã khớp (IN lồng nhau) *(độ khó: dễ)*
- **Bài 5.1** — Chứng khoán chưa có lệnh chờ khớp (NOT EXISTS) *(độ khó: dễ)*
- **Bài 5.1** — Khách hàng chưa giới thiệu ai (bẫy NOT IN với NULL) *(độ khó: trung bình)*
- **Bài 5.1** — Số dư vượt mọi tài khoản của một khách hàng (ALL) *(độ khó: trung bình)*
- **Bài 5.1** — Hồ sơ khách hàng với cột tính từ subquery (SELECT) *(độ khó: trung bình)*
- **Bài 5.1** — Tài sản trung bình theo thành phố (Derived Table) *(độ khó: trung bình)*
- **Bài 5.1** — Khách hàng có tổng tài sản trên trung bình (HAVING) *(độ khó: trung bình)*
- **Bài 5.1** — Tài khoản vượt trung bình cùng loại (Correlated) *(độ khó: trung bình)*
- **Bài 5.2** — Báo cáo phân tầng khách hàng bằng CTE nhiều bước *(độ khó: trung bình)*
- **Bài 5.2** — Giá trị danh mục theo giá thị trường (multiple/nested CTE) *(độ khó: khó)*
- **Bài 5.3** — Phân tích nhiều bước bằng bảng tạm *(độ khó: trung bình)*
- **Bài 5.4** — Tổng hợp: khách hàng nắm giữ lớn nhất từng mã *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: bảng tạm, CTE và đo thời gian chạy

1. Nhìn thấy bảng tạm trong SSMS. Chạy đoạn sau (không đóng cửa sổ):

```
USE hnx_securities;
DROP TABLE IF EXISTS #kh_giau;
SELECT customer_id, SUM(balance) AS tong_so_du
INTO #kh_giau
FROM Accounts
GROUP BY customer_id
HAVING SUM(balance) > 100000000;

SELECT COUNT(*) AS so_kh FROM #kh_giau;
```

Trong Object Explorer mở **Databases → System Databases → tempdb → Temporary Tables** (bấm **Refresh**): bảng `#kh_giau` xuất hiện với đuôi dài kiểu `____000000000012`. Đó là bảng tạm của **phiên (session)** bạn.

2. Kiểm chứng phạm vi của bảng tạm.

1. Mở một cửa sổ truy vấn **mới** (**Ctrl+N**) và chạy `SELECT * FROM #kh_giau;` → lỗi *Invalid object name '#kh_giau'* vì `#` chỉ sống trong phiên tạo ra nó.
2. Thử tạo `##kh_giau_chung` (hai dấu `#`, bảng tạm toàn cục): cửa sổ thứ hai đọc được, và biến mất khi phiên tạo ra nó đóng.
3. Đóng cửa sổ đầu tiên rồi **Refresh** tempdb: bảng `#kh_giau` đã bị xóa tự động.

3. CTE: nhớ dấu chấm phẩy. Câu lệnh đứng ngay trước `WITH` phải kết thúc bằng `;`, nếu không SSMS báo lỗi *Incorrect syntax near the keyword 'with'*:

```
WITH tai_san AS (
  SELECT customer_id, SUM(balance) AS tong
  FROM Accounts
  GROUP BY customer_id
)
SELECT customer_id, tong
FROM tai_san
WHERE tong > (SELECT AVG(tong) FROM tai_san)
ORDER BY tong DESC, customer_id;
```

4. So sánh Subquery, CTE, bảng tạm bằng số liệu. Bật đo trước khi chạy:

```
SET STATISTICS TIME ON;
SET STATISTICS IO ON;
```

Chạy từng cách giải của bài 5.1 rồi xem tab **Messages**: *CPU time*, *elapsed time* và *logical reads*. Với bảng mẫu nhỏ chênh lệch không đáng kể; mục đích là biết cách đọc con số. Tắt bằng `SET STATISTICS TIME OFF; SET STATISTICS IO OFF;`.

Với MySQL Workbench: dùng `CREATE TEMPORARY TABLE ... AS SELECT ...`; bảng tạm chỉ tồn tại trong kết nối đang dùng (mỗi tab thường có kết nối riêng). CTE cần MySQL 8.

Tự kiểm tra: làm lại **bài 5.3** trên SSMS với bảng `#` của riêng bạn rồi nộp lời giải trên Codelab.

Buổi 6 — Thay đổi dữ liệu: INSERT, UPDATE, DELETE

Về mục lục

Năm buổi đầu, chúng ta chỉ **đọc** dữ liệu với `SELECT`. Từ buổi này, mỗi câu lệnh bạn viết có thể **thay đổi dữ liệu thật**: nạp tiền vào tài khoản, đổi trạng thái khách hàng, xóa lệnh giao dịch. Trong công ty chứng khoán, một lệnh `UPDATE` sai có thể làm sai số dư của hàng nghìn khách hàng chỉ trong một phần nghìn giây. Vì vậy buổi học này không chỉ dạy **cú pháp**, mà còn dạy **thói quen an toàn**: kiểm tra trước khi sửa, dùng transaction, sao lưu, và phân quyền.

Bộ dữ liệu mẫu vẫn là `hnx_securities` (xem mục "Bộ dữ liệu mẫu của khóa học" ở Buổi 1). "Hôm nay" cố định là **2026-09-25**. Trên Codelab, **mỗi lần chạy bài (và mỗi lần bấm Chạy thử), hệ thống dùng một bản sao mới của cơ sở dữ liệu**, nên bạn cứ thoải mái thử `INSERT/UPDATE/DELETE`: dữ liệu gốc không bị ảnh hưởng — đúng tinh thần "thao tác trên bảng thử nghiệm" của syllabus. Với các bài loại này, hệ thống **không so sánh kết quả in ra** mà chạy thêm một câu kiểm tra để xem **trạng thái dữ liệu** sau khi script của bạn chạy xong.

> **Nguyên tắc số 1 của buổi học:** trước mỗi `UPDATE` / `DELETE`, hãy viết một câu `SELECT` **cùng điều kiện `WHERE`** để xem chính xác những dòng sẽ bị tác động.

Mục tiêu buổi học

- Sau buổi này bạn có thể viết `INSERT` một dòng, nhiều dòng và `INSERT ... SELECT`; hiểu giá trị mặc định và cột tự tăng (`IDENTITY` / `AUTO_INCREMENT`).
- Sau buổi này bạn có thể viết `UPDATE` (một/nhiều cột, từ subquery) và **nhận biết nguy hiểm** của `UPDATE` thiếu `WHERE`.
- Sau buổi này bạn có thể viết `DELETE` đúng thứ tự khóa ngoại; phân biệt `DELETE`, `TRUNCATE` và biết các cách phục hồi dữ liệu.
- Sau buổi này bạn có thể dùng transaction (`BEGIN` / `COMMIT` / `ROLLBACK`), hiểu **ACID** và ứng dụng để bảo đảm toàn vẹn dữ liệu.
- Sau buổi này bạn có thể áp dụng best practices: backup trước khi thao tác, `SELECT` kiểm tra trước, quản lý quyền truy cập.

Bản đồ buổi học

Buổi học 180 phút: khoảng 26% lý thuyết + demo (~46 phút) và 74% thực hành (~134 phút, gồm cả thời gian chữa bài và kiểm tra lại dữ liệu bằng `SELECT`). Bài tập nằm **xen kẽ** trong bài giảng — học xong phần nào, làm ngay bài của phần đó. Buổi có 15 bài (đa số là thao tác ngắn, kiểm tra bằng trạng thái dữ liệu) nên lý thuyết giảng gọn; bài khó chưa xong trong giờ thì hoàn thành ở nhà.

Phần	Nội dung	Thời lượng
Mục 1	`INSERT` : một dòng, nhiều dòng, `INSERT ... SELECT` , cột tự tăng	9 phút
Thực hành 6.1	Thêm khách hàng, lệnh, giao dịch; nhân bản dữ liệu; nhật ký (4 bài)	31 phút
Mục 2	`UPDATE` : nhiều cột, từ subquery, cảnh báo thiếu `WHERE`	7 phút
Thực hành 6.2	Nạp tiền, kích hoạt khách hàng, nâng loại VIP, đổi loại lệnh (4 bài)	28 phút
Mục 3	`DELETE` , khóa ngoại, `TRUNCATE` , phục hồi dữ liệu	7 phút
Thực hành 6.3	Xóa giao dịch, lệnh cũ, khách hàng không hoạt động, dọn nhật ký (4 bài, có 1 bài khó)	38 phút
Mục 4	Transaction: `BEGIN` / `COMMIT` / `ROLLBACK` , ACID	10 phút
Thực hành 6.4	Chuyển tiền, rollback/commit, khớp lệnh nguyên tử (3 bài, có 1 bài khó)	37 phút
Mục 5	Best practices: backup, `SELECT` trước, phân quyền	5 phút
Tổng kết	Sai lầm thường gặp, hỏi đáp, bài kiểm tra nhanh	8 phút
	Tổng	**180 phút** — lý thuyết + demo ~46 phút, thực hành ~134 phút (74%)

---

1. Lệnh INSERT — thêm dữ liệu

1.1. Cú pháp cơ bản

```
INSERT INTO tên_bảng (cột1, cột2, ...)
VALUES (giá_trị1, giá_trị2, ...);
```

Luôn **liệt kê tên cột** dù có thể bỏ qua. Nếu bạn viết `INSERT INTO Customers VALUES (...)` không có danh sách cột, câu lệnh phụ thuộc vào **thứ tự cột hiện tại** của bảng; khi ai đó thêm cột mới, câu lệnh cũ sẽ hỏng hoặc — tệ hơn — chèn nhầm dữ liệu vào sai cột.

Ví dụ thêm một khách hàng cá nhân mới, rồi kiểm tra ngay bằng `SELECT` :

```
INSERT INTO Customers (customer_id, customer_code, full_name, customer_type,
                       birth_date, city, email, status, created_date)
VALUES (41, 'KH100287', 'Le Minh Quan', 'INDIVIDUAL',
        '1990-05-15', 'Ha Noi', 'le.minh.quan@example.com', 'NEW', '2026-09-25');

SELECT customer_id, customer_code, full_name, status FROM Customers WHERE customer_id = 41;
```

customer_id	customer_code	full_name	status
41	KH100287	Le Minh Quan	NEW

Các cột **không nêu** (`phone` , `referrer_id`) nhận giá trị mặc định của cột — nếu không khai báo `DEFAULT` thì là `NULL` . Riêng cột khai báo `NOT NULL` mà không có `DEFAULT` (như `customer_code` , `full_name` , `customer_type` , `city` , `status` , `created_date`) **bắt buộc** phải được cung cấp giá trị, nếu không câu lệnh báo lỗi.

> **Lưu ý:** hằng chuỗi đặt trong dấu nháy đơn `'...'` ; ngày viết `'yyyy-mm-dd'` ; số **không** đặt trong nháy. Dấu nháy đơn trong dữ liệu phải nhân đôi (`'O''Brien'`).

1.2. INSERT nhiều dòng cùng lúc

Đưa nhiều bộ giá trị vào **một** câu lệnh, phân cách bằng dấu phẩy:

```
INSERT INTO Customers (customer_id, customer_code, full_name, customer_type, birth_date,
        city, email, status, created_date)
VALUES (42, 'KH100294', 'Pham Thu Trang', 'INDIVIDUAL', '1995-11-02', 'Da Nang',
        'pham.thu.trang@example.com', 'NEW', '2026-09-25'),
       (43, 'KH100301', 'Nguyen Duc Thang', 'INDIVIDUAL', '1988-03-27', 'TP.HCM',
        'nguyen.duc.thang@example.com', 'NEW', '2026-09-25');

SELECT customer_id, full_name, city FROM Customers WHERE customer_id >= 42 ORDER BY
        customer_id;
```

Cú pháp này dùng được ở cả SQL Server (từ 2008) và MySQL. Lợi ích: **nhanh hơn** nhiều so với nhiều câu `INSERT` rời rạc, và **nguyên tử** — cả câu lệnh hoặc thành công toàn bộ, hoặc không dòng nào được thêm.

> **SQL Server vs MySQL:** SQL Server cho tối đa **1.000** bộ giá trị trong một `VALUES` ; MySQL giới hạn theo kích thước gói (`max_allowed_packet`), thực tế chèn được hàng nghìn dòng mỗi câu.

1.3. Khóa chính và khóa ngoại khi INSERT

Dữ liệu chèn vào phải thỏa mọi ràng buộc, nếu không toàn bộ câu lệnh bị từ chối:

Vi phạm	SQL Server báo	MySQL báo
Trùng khóa chính (`customer_id` đã tồn tại)	*Violation of PRIMARY KEY constraint*	*Duplicate entry ... for key 'PRIMARY'*
Khóa ngoại không tồn tại (thêm tài khoản cho `customer_id = 999`)	*The INSERT statement conflicted with the FOREIGN KEY constraint*	*Cannot add or update a child row*
Trùng `UNIQUE` (`customer_code` , `account_no`)	*Violation of UNIQUE KEY constraint*	*Duplicate entry*
Thiếu cột `NOT NULL`	*Cannot insert the value NULL into column*	*Field ... doesn't have a default value*

Hệ quả của khóa ngoại: **phải thêm bảng cha trước, bảng con sau** (`Customers` → `Accounts` → `Orders` → `Trades`).

1.4. Thêm dòng cho khách hàng: ghi lệnh và giao dịch

Thêm giao dịch mới thường cần **tra cứu** khóa ngoại thay vì gõ cứng. Ví dụ: khách hàng KH100007 đặt lệnh mua 100 cổ phiếu FPT trên tài khoản CASH đang hoạt động của họ. Ta không nhớ `account_id` , nên dùng `INSERT ... SELECT` để lấy từ `Accounts` :

```
INSERT INTO Orders (order_id, account_id, symbol, side, order_type, quantity, price,
        order_date, status)
SELECT 121, a.account_id, 'FPT', 'BUY', 'LO', 100, 120000, '2026-09-25', 'MATCHED'
FROM Accounts a JOIN Customers c ON c.customer_id = a.customer_id
WHERE c.customer_code = 'KH100007' AND a.account_type = 'CASH' AND a.status = 'ACTIVE';

INSERT INTO Trades (trade_id, order_id, symbol, quantity, price, trade_date)
VALUES (87, 121, 'FPT', 100, 120000, '2026-09-25');

SELECT t.trade_id, o.order_id, o.account_id, t.symbol, t.quantity, t.price
FROM Trades t JOIN Orders o ON o.order_id = t.order_id
WHERE t.trade_id = 87;
```

Chú ý thứ tự: lệnh (`Orders`) phải có **trước** giao dịch (`Trades`) vì `Trades.order_id` là khóa ngoại. Cách này an toàn hơn gõ cứng: nếu khách hàng có tài khoản khác, câu lệnh vẫn chọn đúng tài khoản thỏa điều kiện. Nếu `SELECT` không trả dòng nào (khách hàng không có tài khoản phù hợp), **không có gì được chèn** và cũng không báo lỗi — hãy kiểm tra số dòng bị ảnh hưởng (mục 5).

Vừa học xong `INSERT` một dòng, nhiều dòng, khóa ngoại và tra cứu khóa ngoại bằng `INSERT ... SELECT` , hãy làm bài **6.1a** (thêm khách hàng mới, một dòng và nhiều dòng) và **6.1b** (thêm lệnh và giao dịch mới cho khách hàng).

- **Bài 6.1** — Thêm khách hàng mới (một dòng và nhiều dòng) *(độ khó: dễ)*
- **Bài 6.1** — Thêm lệnh và giao dịch mới cho khách hàng *(độ khó: trung bình)*
- **Bài 6.1** — Nhân bản tài khoản đã đóng vào bảng lưu trữ (INSERT ... SELECT) *(độ khó: trung bình)*
- **Bài 6.1** — Ghi nhật ký với cột tự tăng và giá trị mặc định *(độ khó: trung bình)*

1.5. INSERT ... SELECT: nhân bản dữ liệu từ bảng khác

`INSERT ... SELECT` chèn **kết quả của một truy vấn** vào bảng đích. Dùng để sao lưu, lưu trữ, hoặc đổ dữ liệu trung gian:

```sql
CREATE TABLE AccountArchive (
    account_id  INT PRIMARY KEY,
    customer_id INT NOT NULL,
    balance     DECIMAL(18,2) NOT NULL
);

INSERT INTO AccountArchive (account_id, customer_id, balance)
SELECT account_id, customer_id, balance
FROM Accounts
WHERE status = 'CLOSED';

SELECT account_id, customer_id, balance FROM AccountArchive ORDER BY account_id;
```

Với dữ liệu mẫu, 5 tài khoản `CLOSED` (ID 8, 9, 21, 22, 23) được sao chép. Không dùng `VALUES` khi dùng `SELECT`. Số cột và kiểu dữ liệu ở hai vế phải khớp theo thứ tự.

> **SQL Server vs MySQL:** tạo bảng mới từ kết quả truy vấn *không cần* `CREATE TABLE` trước, nhưng cú pháp khác nhau.

SQL Server:

```sql
SELECT account_id, customer_id, balance INTO AccountArchive2 FROM Accounts WHERE status =
        'CLOSED';
SELECT COUNT(*) AS so_dong FROM AccountArchive2;
```

MySQL:

```sql
CREATE TABLE AccountArchive2 AS SELECT account_id, customer_id, balance FROM Accounts WHERE
        status = 'CLOSED';
SELECT COUNT(*) AS so_dong FROM AccountArchive2;
```

> **Lưu ý:** cách này tiện nhưng **không sao chép khóa chính, index, ràng buộc**; nếu cần đầy đủ, hãy `CREATE TABLE` trước rồi `INSERT ... SELECT`.

1.6. Giá trị mặc định và cột tự tăng (IDENTITY / AUTO_INCREMENT)

Hai cơ chế giúp bạn **không phải cung cấp giá trị** cho một số cột:

- **DEFAULT**: giá trị mặc định khai báo trong bảng (ví dụ `changed_at` mặc định là thời điểm ghi). Bỏ cột khỏi danh sách `INSERT`, hoặc viết `DEFAULT` trong `VALUES`.
- **Cột tự tăng**: hệ thống tự cấp số tiếp theo. SQL Server khai báo `IDENTITY(1,1)`; MySQL khai báo `AUTO_INCREMENT`.

Bảng `AuditLog` của khóa học dùng cả hai (`log_id` tự tăng, `changed_at` mặc định):

```sql
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (5, 'NEW', 'ACTIVE');
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (12, 'NEW', 'ACTIVE');

SELECT log_id, customer_id, old_status, new_status FROM AuditLog ORDER BY log_id;
```

log_id	customer_id	old_status	new_status
1	5	NEW	ACTIVE
2	12	NEW	ACTIVE

Những điều cần biết về cột tự tăng:

1. **Không được chỉ định** giá trị cho cột `IDENTITY` (SQL Server sẽ báo lỗi) trừ khi bật `SET IDENTITY_INSERT AuditLog ON`. Sau khi chèn tay `log_id = 100`, dòng kế tiếp tự nhận 101.
2. **Số đã cấp không được tái sử dụng**: nếu bạn `DELETE` toàn bộ bảng rồi chèn tiếp, `log_id` **tiếp nối** (ví dụ 3) chứ không quay lại 1. Nếu chèn trong transaction rồi `ROLLBACK`, số đó cũng **bị "cháy"** (ở cả hai hệ, dòng chèn kế tiếp sẽ có `log_id = 2`). Vì vậy **đừng dùng cột tự tăng làm số liên tục kế toán**.
3. Lấy giá trị vừa được cấp: SQL Server `SELECT SCOPE_IDENTITY();`, MySQL `SELECT LAST_INSERT_ID();` (dùng để chèn tiếp vào bảng con).

> **Ví dụ nghiệp vụ:** `AuditLog` là "hộp đen" của phòng Tuân thủ: mỗi lần trạng thái khách hàng đổi, một dòng mới được ghi kèm thời điểm. Nhờ `log_id` tự tăng và `changed_at` mặc định, ứng dụng chỉ cần gửi 3 giá trị — ít lỗi hơn và không thể "quên" ghi thời điểm.

Vừa học xong `INSERT ... SELECT` để nhân bản dữ liệu, giá trị mặc định và cột tự tăng, hãy làm bài **6.1c** (nhân bản tài khoản đã đóng vào bảng lưu trữ) và **6.1d** (ghi nhật ký với cột tự tăng và giá trị mặc định).

- **Bài 6.1** — Thêm khách hàng mới (một dòng và nhiều dòng) *(độ khó: dễ)*
- **Bài 6.1** — Thêm lệnh và giao dịch mới cho khách hàng *(độ khó: trung bình)*
- **Bài 6.1** — Nhân bản tài khoản đã đóng vào bảng lưu trữ (INSERT ... SELECT) *(độ khó: trung bình)*
- **Bài 6.1** — Ghi nhật ký với cột tự tăng và giá trị mặc định *(độ khó: trung bình)*

---

2. Lệnh UPDATE — sửa dữ liệu

2.1. Cú pháp

```sql
UPDATE tên_bảng
SET cột1 = giá_trị1, cột2 = giá_trị2
WHERE điều_kiện;
```

Vế phải của `SET` có thể là hằng, biểu thức dựa trên chính cột đó, hoặc subquery. Ví dụ **nạp 5.000.000** vào tài khoản `ACC00003`:

```sql
UPDATE Accounts
SET balance = balance + 5000000
WHERE account_no = 'ACC00003';

SELECT account_no, balance FROM Accounts WHERE account_no = 'ACC00003';
```

Số dư tài khoản này đang là 32.000.000 → 37.000.000. Lưu ý dùng `balance = balance + số tiền` (cộng dồn) thay vì gán cứng số mới: nếu có giao dịch khác cùng lúc, cách cộng dồn không ghi đè mất thay đổi của người khác.

2.2. UPDATE nhiều cột cùng lúc

Đưa nhiều phép gán trong **một** `SET`, cách nhau dấu phẩy. Ví dụ chuyển lệnh thị trường (MP) chưa khớp thành lệnh giới hạn (LO) với giá đóng cửa 2026-09-25 — vừa đổi hai cột, vừa **lấy giá từ subquery**:

```sql
UPDATE Orders
SET order_type = 'LO',
    price = (SELECT m.close_price FROM MarketPrices m
             WHERE m.symbol = Orders.symbol AND m.trade_date = '2026-09-25')
WHERE order_type = 'MP' AND status = 'PENDING';

SELECT order_id, symbol, order_type, price FROM Orders WHERE order_id IN (6, 34, 62, 104,
        118) ORDER BY order_id;
```

Câu lệnh này chuyển 5 lệnh (ID 6, 34, 62, 104, 118). Hiểu quy tắc của SQL: **toàn bộ vế phải được tính dựa trên giá trị cũ của dòng**, rồi mới gán.

> **SQL Server vs MySQL:** vì quy tắc trên, `SET a = b, b = a` sẽ **hoán đổi** giá trị trong SQL Server, còn MySQL xử lý gán từ trái sang phải nên kết quả khác! Tránh viết những câu như vậy.

Vừa học xong `UPDATE` một cột (cộng dồn) và nhiều cột với giá trị lấy từ subquery, hãy làm bài **6.2a** (nạp tiền vào tài khoản — nhớ `WHERE`) và **6.2d** (chuyển lệnh MP chờ khớp thành LO theo giá đóng cửa).

- **Bài 6.2** — Nạp tiền vào tài khoản (UPDATE có WHERE) *(độ khó: dễ)*
- **Bài 6.2** — Chuyển khách hàng mới sang trạng thái kích hoạt *(độ khó: dễ)*
- **Bài 6.2** — Nâng loại khách hàng lên VIP theo tổng tài sản (UPDATE từ subquery) *(độ khó: trung bình)*
- **Bài 6.2** — Cập nhật nhiều cột: chuyển lệnh MP chờ khớp thành LO theo giá đóng cửa *(độ khó: trung bình)*

2.3. Cập nhật có điều kiện theo dữ liệu bảng khác

Thay đổi loại khách hàng: khách hàng có tổng tài sản **lớn hơn 5 tỷ** thành `VIP`:

```sql
UPDATE Customers
SET customer_type = 'VIP'
WHERE customer_id IN (SELECT customer_id
                      FROM Accounts
                      GROUP BY customer_id
                      HAVING SUM(balance) > 5000000000);

SELECT customer_id, full_name, customer_type FROM Customers WHERE customer_type = 'VIP'
        ORDER BY customer_id;
```

Với dữ liệu mẫu, 3 khách hàng được nâng: ID 2 (8,64 tỷ), 17 (12,07 tỷ), 40 (9,5 tỷ). Chú ý so sánh `>` (**không** phải `>=`): khách hàng có đúng 5 tỷ không đủ điều kiện. Cách viết `IN (subquery)` (Buổi 5) chạy giống nhau trên cả hai hệ.

Khi muốn **nối bảng** (JOIN) trong `UPDATE`, cú pháp **khác nhau**. Ví dụ khóa các tài khoản `ACTIVE` của khách hàng `INACTIVE`:

SQL Server — `UPDATE alias SET ... FROM bảng alias JOIN ...`:

```sql
UPDATE a SET a.status = 'LOCKED'
FROM Accounts a JOIN Customers c ON c.customer_id = a.customer_id
WHERE c.status = 'INACTIVE' AND a.status = 'ACTIVE';

SELECT COUNT(*) AS so_tai_khoan_bi_khoa FROM Accounts WHERE status = 'LOCKED';
```

MySQL — `UPDATE bảng JOIN ... SET ...`:

```sql
UPDATE Accounts a JOIN Customers c ON c.customer_id = a.customer_id
SET a.status = 'LOCKED'
WHERE c.status = 'INACTIVE' AND a.status = 'ACTIVE';

SELECT COUNT(*) AS so_tai_khoan_bi_khoa FROM Accounts WHERE status = 'LOCKED';
```

Cả hai khóa 7 tài khoản `ACTIVE` của các khách hàng `INACTIVE` (17, 23, 36). Cách dùng `IN (subquery)` như trên **không phụ thuộc dialect**, nên là lựa chọn dễ mang đi giữa hai hệ.

2.4. Chuyển trạng thái hàng loạt

```sql
UPDATE Customers SET status = 'ACTIVE' WHERE status = 'NEW';
```

Bốn khách hàng đang `NEW` (ID 5, 12, 19, 26) chuyển thành `ACTIVE`; 5 khách hàng `INACTIVE` không bị đụng tới. Trước khi chạy, hãy đếm: `SELECT COUNT(*) FROM Customers WHERE status = 'NEW';` (kết quả 4) và **so khớp** số dòng bị ảnh hưởng sau khi chạy.

Vừa học xong `UPDATE` theo dữ liệu bảng khác và chuyển trạng thái hàng loạt, hãy làm bài **6.2b** (chuyển khách hàng mới sang trạng thái kích hoạt) và **6.2c** (nâng loại khách hàng lên VIP theo tổng tài sản).

- **Bài 6.2** — Nạp tiền vào tài khoản (UPDATE có WHERE) *(độ khó: dễ)*
- **Bài 6.2** — Chuyển khách hàng mới sang trạng thái kích hoạt *(độ khó: dễ)*
- **Bài 6.2** — Nâng loại khách hàng lên VIP theo tổng tài sản (UPDATE từ subquery) *(độ khó: trung bình)*
- **Bài 6.2** — Cập nhật nhiều cột: chuyển lệnh MP chờ khớp thành LO theo giá đóng cửa *(độ khó: trung bình)*

2.5. CẢNH BÁO: UPDATE không có WHERE

Hãy xem chuyện gì xảy ra nếu **quên** `WHERE`:

```
UPDATE Accounts SET balance = 0;      -- KHÔNG CÓ WHERE!

SELECT SUM(balance) AS tong_so_du FROM Accounts;
```

Kết quả `0.00`: câu lệnh **đặt số dư của cả 93 tài khoản về 0**. Không có cảnh báo, không hỏi lại. Đây là lỗi vận hành nổi tiếng: quên `WHERE`, hoặc bôi đen chỉ một phần câu lệnh trong SSMS rồi bấm chạy (chỉ chạy đoạn bôi đen, mất `WHERE`).

> **Cảnh báo:** trên hệ thống thật, câu lệnh trên xóa sạch số dư của toàn công ty trong tích tắc. Bài **6.2a** được chấm bằng cả **tổng số dư toàn bảng** — nếu bạn quên `WHERE`, kết quả sai ngay lập tức.

Cách phòng tránh:

1. **Viết `WHERE` trước**, gõ `UPDATE` / `SET` sau; hoặc viết dạng `SELECT` trước rồi đổi thành `UPDATE`.
2. **Dùng transaction** (mục 4): kiểm tra số dòng bị ảnh hưởng, sai thì `ROLLBACK`.
3. MySQL Workbench có chế độ **Safe Updates** (`SET sql_safe_updates = 1;`) từ chối `UPDATE` / `DELETE` không có `WHERE` dùng khóa; nên bật trên môi trường làm việc.

Một lỗi khác: viết `WHERE cột = NULL`. `NULL` không bằng gì cả (kể cả `NULL`); phải dùng `WHERE cột IS NULL` và `SET cột = NULL` để gán.

---

3. Lệnh DELETE — xóa dữ liệu

3.1. Cú pháp

```
DELETE FROM tên_bảng
WHERE điều_kiện;
```

Ví dụ xóa toàn bộ **giao dịch (`Trades`) của khách hàng KH100014**. Bảng `Trades` không có cột `customer_id`, nên phải đi qua chuỗi `Trades → Orders → Accounts → Customers` bằng subquery. Theo nguyên tắc số 1, ta **xem trước** rồi mới xóa:

```sql
-- Bước 1: xem trước bao nhiêu dòng sẽ bị xóa
SELECT COUNT(*) AS so_dong_se_xoa FROM Trades
WHERE order_id IN (SELECT o.order_id FROM Orders o
                   JOIN Accounts a ON a.account_id = o.account_id
                   JOIN Customers c ON c.customer_id = a.customer_id
                   WHERE c.customer_code = 'KH100014');
```

Kết quả là **4**. Bước 2 — xóa (cùng điều kiện), rồi đếm lại toàn bảng:

```sql
DELETE FROM Trades
WHERE order_id IN (SELECT o.order_id FROM Orders o
                   JOIN Accounts a ON a.account_id = o.account_id
                   JOIN Customers c ON c.customer_id = a.customer_id
                   WHERE c.customer_code = 'KH100014');

SELECT COUNT(*) AS so_giao_dich_con_lai FROM Trades;
```

Trước khi xóa `Trades` có 86 dòng, sau khi xóa còn 82. Lưu ý: `DELETE` chỉ xóa **bảng đích**; `Orders` của khách hàng này vẫn còn.

3.2. Xóa và ràng buộc khóa ngoại

Khóa ngoại **bảo vệ** dữ liệu: bạn không thể xóa dòng cha nếu còn dòng con tham chiếu:

```sql
DELETE FROM Customers WHERE customer_id = 2;
-- SQL Server: The DELETE statement conflicted with the REFERENCE constraint ...
-- MySQL:      Cannot delete or update a parent row: a foreign key constraint fails
```

Quy tắc: **xóa từ bảng con lên bảng cha**. Với hệ thống chứng khoán của khóa học:

`Trades` → `Orders` → `Portfolio` → `Accounts` → `CustomerStats` → `Customers`

Ví dụ bài **6.3b**: xóa các lệnh `CANCELLED` trước 2026-07-01 (7 lệnh). Nếu có lệnh còn giao dịch đi kèm thì phải xóa giao dịch trước:

```sql
DELETE FROM Trades WHERE order_id IN (SELECT order_id FROM Orders WHERE status = 'CANCELLED'
        AND order_date < '2026-07-01');
DELETE FROM Orders WHERE status = 'CANCELLED' AND order_date < '2026-07-01';
```

> **Cảnh báo:** thay vì xóa thủ công từng tầng, cơ sở dữ liệu có thể khai báo `ON DELETE CASCADE` (xóa dây chuyền) — thuận tiện nhưng **nguy hiểm** vì một câu `DELETE` có thể xóa lan rất rộng; hệ thống tài chính thường **không** bật cascade cho dữ liệu giao dịch.

Vừa học xong `DELETE` với `WHERE`, subquery và thứ tự khóa ngoại, hãy làm bài **6.3a** (xóa giao dịch của một khách hàng cụ thể) và **6.3b** (xóa lệnh đã hủy cũ, tôn trọng khóa ngoại).

- **Bài 6.3** — Xóa giao dịch của một khách hàng cụ thể *(độ khó: trung bình)*
- **Bài 6.3** — Xóa lệnh đã hủy cũ (tôn trọng khóa ngoại) *(độ khó: trung bình)*
- **Bài 6.3** — Xóa khách hàng không hoạt động và toàn bộ dữ liệu liên quan *(độ khó: khó)*
- **Bài 6.3** — DELETE và TRUNCATE: dọn bảng nhật ký và đánh số lại *(độ khó: trung bình)*

3.3. Xóa khách hàng không hoạt động (bài toán tổng hợp)

Bài **6.3c** yêu cầu xóa các khách hàng `INACTIVE` và không có giao dịch từ 2025-09-25 (12 tháng), cùng toàn bộ dữ liệu phụ thuộc. Mẫu giải quyết chuẩn: **chốt danh sách trước bằng bảng tạm** (Buổi 5), rồi xóa từ dưới lên.

SQL Server:

```
SELECT c.customer_id INTO #del
FROM Customers c
WHERE c.status = 'INACTIVE'
  AND NOT EXISTS (SELECT 1 FROM Trades t JOIN Orders o ON o.order_id = t.order_id
                  JOIN Accounts a ON a.account_id = o.account_id
                  WHERE a.customer_id = c.customer_id AND t.trade_date >= '2025-09-25');
```

MySQL — `CREATE TEMPORARY TABLE del AS SELECT c.customer_id FROM Customers c WHERE ...` (cùng điều kiện). Sau đó, ở cả hai hệ, xóa từ bảng con lên bảng cha (dưới đây viết cho SQL Server với `#del`; MySQL đổi `#del` thành `del`):

```
DELETE FROM Trades    WHERE order_id   IN (SELECT o.order_id FROM Orders o JOIN Accounts a
        ON a.account_id = o.account_id
                                          WHERE a.customer_id IN (SELECT customer_id FROM
        #del));
DELETE FROM Orders    WHERE account_id IN (SELECT account_id FROM Accounts WHERE customer_id
        IN (SELECT customer_id FROM #del));
DELETE FROM Portfolio WHERE account_id IN (SELECT account_id FROM Accounts WHERE customer_id
        IN (SELECT customer_id FROM #del));
DELETE FROM Accounts  WHERE customer_id IN (SELECT customer_id FROM #del);
DELETE FROM CustomerStats WHERE customer_id IN (SELECT customer_id FROM #del);
DELETE FROM Customers WHERE customer_id IN (SELECT customer_id FROM #del);
```

Vì sao phải chốt danh sách trước? Tiêu chí "không có giao dịch trong 12 tháng" được tính qua chuỗi nối `Trades → Orders → Accounts`; khi bạn xóa dần từng tầng thì chuỗi nối này bị đứt, **không thể tính lại** tiêu chí ở các câu `DELETE` sau. Bảng tạm `#del` giữ cố định danh sách gốc cho mọi câu lệnh. Với dữ liệu mẫu, khách hàng 4 và 9 bị xóa; khách hàng 17, 23, 36 dù `INACTIVE` vẫn được giữ vì có giao dịch gần đây. Sau đó `Customers` còn 38 dòng, `Accounts` còn 88, `Orders` giữ 120, `Trades` giữ 86.

3.4. DELETE, TRUNCATE và DROP

	`DELETE`	`TRUNCATE TABLE`	`DROP TABLE`
Xóa cái gì	các dòng thỏa `WHERE` (hoặc tất cả)	**tất cả** dòng	cả bảng (cấu trúc + dữ liệu)
Có `WHERE`	có	**không**	không
Cột tự tăng	**không** đặt lại	**đặt lại** về giá trị khởi đầu	mất cùng bảng
Tốc độ (bảng lớn)	chậm (ghi log từng dòng)	rất nhanh	rất nhanh
Kích hoạt trigger	có	không	không
Bảng có khóa ngoại tham chiếu tới	được (nếu không còn dòng con)	**không được** (cả hai hệ)	không được
Rollback trong transaction	có	SQL Server: **có**; MySQL: **không** (tự động COMMIT)	tương tự

Hãy tự kiểm chứng sự khác biệt về cột tự tăng. Với `TRUNCATE`, dòng chèn sau cùng có `log_id = 1`:

```
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (5, 'NEW', 'ACTIVE');
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (12, 'NEW', 'ACTIVE');

TRUNCATE TABLE AuditLog;
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (19, 'NEW', 'ACTIVE');

SELECT log_id, customer_id FROM AuditLog;
```

Còn nếu thay `TRUNCATE TABLE AuditLog;` bằng `DELETE FROM AuditLog;`, dòng mới mang số **tiếp nối**:

```
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (5, 'NEW', 'ACTIVE');
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (12, 'NEW', 'ACTIVE');

DELETE FROM AuditLog;
INSERT INTO AuditLog (customer_id, old_status, new_status) VALUES (19, 'NEW', 'ACTIVE');

SELECT log_id, customer_id FROM AuditLog;
```

Kết quả `log_id = 3` (tiếp nối) thay vì 1.

Vừa học xong xóa nhiều tầng bằng bảng tạm (mục 3.3) và sự khác nhau giữa `DELETE` với `TRUNCATE`, hãy làm bài **6.3c** (xóa khách hàng không hoạt động và toàn bộ dữ liệu liên quan) và **6.3d** (dọn bảng nhật ký, đánh số lại).

- **Bài 6.3** — Xóa giao dịch của một khách hàng cụ thể *(độ khó: trung bình)*
- **Bài 6.3** — Xóa lệnh đã hủy cũ (tôn trọng khóa ngoại) *(độ khó: trung bình)*
- **Bài 6.3** — Xóa khách hàng không hoạt động và toàn bộ dữ liệu liên quan *(độ khó: khó)*
- **Bài 6.3** — DELETE và TRUNCATE: dọn bảng nhật ký và đánh số lại *(độ khó: trung bình)*

> **SQL Server vs MySQL:** `TRUNCATE`, `CREATE`, `DROP`, `ALTER` trong MySQL là lệnh **DDL** và gây **implicit commit** — mọi thay đổi đang treo trong transaction sẽ bị chốt ngay, không thể `ROLLBACK`. Không đặt `TRUNCATE` vào giữa một transaction quan trọng.

3.5. Phục hồi dữ liệu sau DELETE

Một khi `DELETE` đã `COMMIT`, dữ liệu **không tự quay lại**. Các lớp bảo vệ, từ nhẹ đến nặng:

Mức	Cách	Điều kiện
1	**Chưa COMMIT**: `ROLLBACK` (mục 4)	cứu hộ tức thì; lý do nên chạy thao tác nguy hiểm trong transaction
2	**Chụp lại dòng bị xóa trước khi xóa**: `INSERT ... SELECT` sang bảng lưu trữ (như 6.1c); SQL Server có `OUTPUT deleted.*` để ghi dòng vừa xóa vào bảng khác trong cùng câu lệnh	chuẩn bị trước khi xóa
3	**Xóa mềm (soft delete)**: đặt `status = 'DELETED'` hoặc `is_deleted = 1` và lọc khi truy vấn	thiết kế bảng từ đầu; thường dùng với dữ liệu tài chính vì luật yêu cầu lưu vết
4	**Khôi phục từ backup**: SQL Server `RESTORE DATABASE ... WITH STOPAT = 'thời điểm'` (bản full + log); MySQL khôi phục `mysqldump` rồi **áp lại binary log** đến trước thời điểm sai	việc của DBA, cần chuẩn bị từ trước (mục 5)

---

4. Transaction (giao dịch)

4.1. Vấn đề cần giải quyết

Chuyển 10.000.000 từ tài khoản A sang B gồm **hai** `UPDATE`. Nếu máy chủ mất điện sau lệnh trừ mà chưa kịp lệnh cộng, khách hàng **mất tiền**. Ta muốn hai việc **hoặc cùng thành công, hoặc cùng không xảy ra**. **Transaction** gom nhiều lệnh thành một **đơn vị công việc không thể chia nhỏ**.

4.2. BEGIN, COMMIT, ROLLBACK

Thao tác	SQL Server	MySQL
Bắt đầu	`BEGIN TRANSACTION`	`START TRANSACTION` (hoặc `BEGIN`)
Chốt (lưu vĩnh viễn)	`COMMIT TRANSACTION`	`COMMIT`
Hủy (quay về trước khi bắt đầu)	`ROLLBACK TRANSACTION`	`ROLLBACK`

Hình dưới đây mô tả vòng đời của một transaction: mở bằng `BEGIN`, chạy các lệnh, rồi kết thúc bằng `COMMIT` (chốt tất cả) hoặc `ROLLBACK` (hủy tất cả); có thể đặt `SAVEPOINT` để hủy một phần.

Hình minh họa: Luồng của một transaction (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: đi từ trên xuống. Mọi thay đổi giữa `BEGIN` và `COMMIT` chỉ là **tạm thời** với các phiên khác. Nếu mọi lệnh thành công và kết quả đúng kỳ vọng, nhánh `COMMIT` biến chúng thành **vĩnh viễn**. Nếu có lỗi hoặc kết quả sai, nhánh `ROLLBACK` đưa dữ liệu về **đúng trạng thái trước `BEGIN`**. Nếu có điểm lưu, `ROLLBACK` đến điểm lưu chỉ hủy phần sau điểm đó.

Chuyển tiền thành công — bài **6.4a**. Sau khi `COMMIT`, số dư mới được giữ lại:

SQL Server:

```sql
BEGIN TRANSACTION;
UPDATE Accounts SET balance = balance - 10000000 WHERE account_id = 1;
UPDATE Accounts SET balance = balance + 10000000 WHERE account_id = 2;
COMMIT TRANSACTION;

SELECT account_id, balance FROM Accounts WHERE account_id IN (1, 2) ORDER BY account_id;
```

MySQL:

```sql
START TRANSACTION;
UPDATE Accounts SET balance = balance - 10000000 WHERE account_id = 1;
UPDATE Accounts SET balance = balance + 10000000 WHERE account_id = 2;
COMMIT;

SELECT account_id, balance FROM Accounts WHERE account_id IN (1, 2) ORDER BY account_id;
```

Tài khoản 1: 67.000.000 → 57.000.000; tài khoản 2: 8.000.000 → 18.000.000; **tổng số dư không đổi** (tính bảo toàn).

Hủy giữa chừng bằng `ROLLBACK` — số dư trở về như cũ:

```sql
BEGIN TRANSACTION;
UPDATE Accounts SET balance = 0 WHERE account_id = 1;
ROLLBACK TRANSACTION;

SELECT balance FROM Accounts WHERE account_id = 1;
```

```sql
START TRANSACTION;
UPDATE Accounts SET balance = 0 WHERE account_id = 1;
ROLLBACK;

SELECT balance FROM Accounts WHERE account_id = 1;
```

Kết quả `67000000.00` (không đổi).

> **Lưu ý:** cả hai hệ chạy ở chế độ **autocommit** — *mỗi câu lệnh riêng lẻ tự COMMIT ngay*. Bạn chỉ có khả năng `ROLLBACK` khi đã mở transaction tường minh. Trên Codelab, runner cũng chạy ở chế độ autocommit, vì vậy `BEGIN/COMMIT/ROLLBACK` trong script của bạn có tác dụng thật.

Bài **6.4b** cho bạn thử cả hai kết cục trên bảng `Orders` (kiểm tra cơ chế xóa lệnh): transaction thử nghiệm xóa **toàn bộ** lệnh `PENDING` rồi `ROLLBACK` (17 lệnh vẫn còn), sau đó transaction thật chỉ xóa lệnh `PENDING` trước 2026-09-01 rồi `COMMIT` (8 lệnh bị xóa, còn 9).

4.3. Điểm lưu (savepoint) và xử lý lỗi

Savepoint cho phép hủy một phần: chuyển 10.000.000 nhưng "gõ nhầm" bước cộng — hủy riêng bước sai mà vẫn giữ bước trước.

SQL Server — `SAVE TRANSACTION` / `ROLLBACK TRANSACTION tên`:

```
BEGIN TRANSACTION;
UPDATE Accounts SET balance = balance - 10000000 WHERE account_id = 1;
SAVE TRANSACTION diem1;
UPDATE Accounts SET balance = balance + 999 WHERE account_id = 2;   -- bước sai
ROLLBACK TRANSACTION diem1;                                         -- chỉ hủy phần sau
        diem1
COMMIT TRANSACTION;

SELECT account_id, balance FROM Accounts WHERE account_id IN (1, 2) ORDER BY account_id;
```

MySQL — `SAVEPOINT` / `ROLLBACK TO SAVEPOINT`:

```
START TRANSACTION;
UPDATE Accounts SET balance = balance - 10000000 WHERE account_id = 1;
SAVEPOINT diem1;
UPDATE Accounts SET balance = balance + 999 WHERE account_id = 2;   -- bước sai
ROLLBACK TO SAVEPOINT diem1;                                        -- chỉ hủy phần sau
        diem1
COMMIT;

SELECT account_id, balance FROM Accounts WHERE account_id IN (1, 2) ORDER BY account_id;
```

Kết quả: tài khoản 1 còn 57.000.000 (bước trừ được giữ), tài khoản 2 vẫn 8.000.000 (bước sai bị hủy).

- **SQL Server** thường viết trong `BEGIN TRY ... END TRY BEGIN CATCH ROLLBACK ... END CATCH` và bật `SET XACT_ABORT ON` để **bất kỳ lỗi nào cũng tự hủy** toàn bộ transaction. **MySQL** xử lý lỗi bằng `DECLARE EXIT HANDLER FOR SQLEXCEPTION` trong stored procedure (Buổi sau).
- Không để transaction **mở quá lâu**: nó giữ **khóa (lock)** trên các dòng đã sửa, người khác phải chờ. Giữ transaction ngắn, chỉ chứa các lệnh cần thiết.

> **Mẹo:** mô phỏng hai phiên chờ nhau/deadlock không chấm tự động được trên Codelab. Bạn có thể tự thử trong SSMS bằng hai cửa sổ Query: cửa sổ 1 chạy `BEGIN TRANSACTION; UPDATE Accounts SET balance = balance + 1 WHERE account_id = 1;` (chưa COMMIT); cửa sổ 2 chạy `SELECT balance FROM Accounts WHERE account_id = 1;` — truy vấn sẽ **treo chờ** cho đến khi cửa sổ 1 `COMMIT` hoặc `ROLLBACK`. Nhớ đóng transaction dở dang sau khi thử.

4.4. ACID — bốn tính chất của transaction

Tính chất	Ý nghĩa	Ví dụ ngành chứng khoán
Atomicity (nguyên tử)	tất cả hoặc không gì cả	khớp lệnh: cập nhật lệnh + ghi giao dịch + trừ tiền + cộng cổ phiếu — thiếu một bước là sai
Consistency (nhất quán)	dữ liệu luôn thỏa ràng buộc và quy tắc nghiệp vụ	tổng số dư bảo toàn; không ghi giao dịch cho lệnh không tồn tại
Isolation (cô lập)	transaction đồng thời không "nhìn thấy" trạng thái dở dang của nhau	hai nhân viên cùng sửa số dư một tài khoản không làm mất cập nhật của nhau
Durability (bền vững)	đã `COMMIT` thì không mất, kể cả khi sập nguồn	lệnh đã khớp phải còn sau khi khởi động lại máy chủ

> **SQL Server vs MySQL:** mức cô lập mặc định của SQL Server là `READ COMMITTED`; của MySQL (InnoDB) là `REPEATABLE READ`. Bạn sẽ gặp lại khi học về hiệu năng và khóa.

4.5. Ứng dụng: khớp lệnh nguyên tử

Bài **6.4c** gói bốn thao tác của việc khớp lệnh mua 27 (tài khoản 62 mua 200 VCS giá 60.000) vào **một** transaction: đổi trạng thái lệnh, ghi giao dịch, trừ 12.000.000, tăng danh mục từ 100 lên 300 cổ phiếu. Mọi số liệu lấy bằng subquery từ chính lệnh 27. Ví dụ SQL Server (MySQL: đổi `BEGIN TRANSACTION` thành `START TRANSACTION`, `COMMIT TRANSACTION` thành `COMMIT`):

```
BEGIN TRANSACTION;
UPDATE Orders SET status = 'MATCHED' WHERE order_id = 27;

INSERT INTO Trades (trade_id, order_id, symbol, quantity, price, trade_date)
SELECT 87, order_id, symbol, quantity, price, '2026-09-25' FROM Orders WHERE order_id = 27;

UPDATE Accounts
SET balance = balance - (SELECT quantity * price FROM Orders WHERE order_id = 27)
WHERE account_id = (SELECT account_id FROM Orders WHERE order_id = 27);

UPDATE Portfolio
SET quantity = quantity + (SELECT quantity FROM Orders WHERE order_id = 27)
WHERE account_id = (SELECT account_id FROM Orders WHERE order_id = 27)
  AND symbol = (SELECT symbol FROM Orders WHERE order_id = 27);
COMMIT TRANSACTION;

SELECT (SELECT status FROM Orders WHERE order_id = 27) AS order_status,
       (SELECT balance FROM Accounts WHERE account_id = 62) AS balance,
       (SELECT quantity FROM Portfolio WHERE account_id = 62 AND symbol = 'VCS') AS
        vcs_quantity;
```

Cùng ví dụ trên MySQL:

```
START TRANSACTION;
UPDATE Orders SET status = 'MATCHED' WHERE order_id = 27;

INSERT INTO Trades (trade_id, order_id, symbol, quantity, price, trade_date)
SELECT 87, order_id, symbol, quantity, price, '2026-09-25' FROM Orders WHERE order_id = 27;

UPDATE Accounts
SET balance = balance - (SELECT quantity * price FROM Orders WHERE order_id = 27)
WHERE account_id = (SELECT account_id FROM Orders WHERE order_id = 27);

UPDATE Portfolio
SET quantity = quantity + (SELECT quantity FROM Orders WHERE order_id = 27)
WHERE account_id = (SELECT account_id FROM Orders WHERE order_id = 27)
  AND symbol = (SELECT symbol FROM Orders WHERE order_id = 27);
COMMIT;

SELECT (SELECT status FROM Orders WHERE order_id = 27) AS order_status,
       (SELECT balance FROM Accounts WHERE account_id = 62) AS balance,
       (SELECT quantity FROM Portfolio WHERE account_id = 62 AND symbol = 'VCS') AS
        vcs_quantity;
```

> **Lưu ý:** hãy để ý một điểm tinh tế: bước 1 đổi trạng thái lệnh trước, nhưng các bước sau vẫn đọc được `quantity`, `price`, `account_id` từ cùng dòng đó (chỉ `status` bị đổi). Nếu thay vì `UPDATE Orders` bạn đổi giá hay số lượng ở bước đầu thì các bước sau sẽ đọc phải giá trị mới — luôn đọc dữ liệu nguồn **trước** khi sửa nó.

Vừa học xong transaction và ACID, hãy làm 3 bài **6.4a** → **6.4c**: chuyển tiền giữa hai tài khoản, thử `ROLLBACK` / `COMMIT` khi xóa lệnh chờ khớp, và khớp lệnh mua trong một transaction nguyên tử.

- **Bài 6.4** — Chuyển tiền giữa hai tài khoản bằng transaction *(độ khó: trung bình)*
- **Bài 6.4** — ROLLBACK và COMMIT: thử xóa lệnh chờ khớp *(độ khó: trung bình)*
- **Bài 6.4** — Khớp lệnh mua trong một transaction (ACID) *(độ khó: khó)*

---

5. Best practices và bảo mật

5.1. Backup trước khi thao tác

- Trước đợt thay đổi lớn, có **bản sao lưu đã kiểm chứng phục hồi được**.
- Cho các thay đổi nhỏ: **sao lưu tại chỗ** phần dữ liệu sắp sửa bằng `INSERT ... SELECT` vào bảng lưu trữ (bài 6.1c), hoặc `SELECT ... INTO backup_20260925 FROM ...`.
- Backup chỉ có giá trị khi **đã thử restore**. Hệ thống thật của công ty (xem tài liệu vận hành) có lịch backup hằng ngày; DBA chịu trách nhiệm, còn bạn chịu trách nhiệm **không làm sai** trước tiên.

	SQL Server	**MySQL**
Sao lưu toàn bộ	`BACKUP DATABASE hnx_securities TO DISK = 'D:\backup\hnx.bak'`	`mysqldump hnx_securities > hnx.sql`
Sao lưu thay đổi	differential + log backup	binary log
Phục hồi	`RESTORE DATABASE ...` (`WITH STOPAT` để về đúng thời điểm)	nạp lại file dump, áp binary log đến thời điểm cần

5.2. SELECT trước UPDATE/DELETE

Quy trình 5 bước cho mọi thay đổi dữ liệu trên môi trường thật:

1. Viết `SELECT` với đúng `WHERE` → xem các dòng sẽ bị tác động, **đếm** số dòng.
2. Mở transaction (`BEGIN TRANSACTION` / `START TRANSACTION`).
3. Đổi `SELECT` thành `UPDATE` / `DELETE` (giữ nguyên `WHERE`).
4. Kiểm tra số dòng bị ảnh hưởng, rồi `SELECT` lại để xem kết quả.
5. Đúng như kỳ vọng → `COMMIT`; khác kỳ vọng → `ROLLBACK`.

Bước 4 dùng hàm khác nhau ở hai hệ. Ví dụ cộng 1 đồng cho hai tài khoản và đọc số dòng bị ảnh hưởng (kết quả `2`):

```sql
UPDATE Accounts SET balance = balance + 1 WHERE account_id IN (1, 2);
SELECT @@ROWCOUNT AS so_dong_bi_anh_huong;
```

```sql
UPDATE Accounts SET balance = balance + 1 WHERE account_id IN (1, 2);
SELECT ROW_COUNT() AS so_dong_bi_anh_huong;
```

> **Mẹo:** nếu con số này **khác** số bạn đếm ở bước 1 (ví dụ kỳ vọng 4 dòng nhưng ra 4.000), hãy `ROLLBACK` ngay. Thêm: chạy lần đầu trên **bản sao/bảng thử nghiệm**; không bao giờ thử trên production; **luôn liệt kê cột** trong `INSERT`; đọc kỹ lại **toàn bộ** câu lệnh (đúng bảng, đúng `WHERE`, đúng cơ sở dữ liệu) trước khi bấm chạy.

5.3. Quản lý quyền truy cập

Nguyên tắc **đặc quyền tối thiểu** (least privilege): mỗi người chỉ có quyền đủ để làm việc của mình.

Vai trò	Quyền điển hình
Nhà phân tích / báo cáo	chỉ `SELECT`
Nhân viên vận hành back-office	`SELECT`, `INSERT`, `UPDATE` trên một số bảng; **không** `DELETE` / `TRUNCATE`
DBA	toàn quyền, thao tác có ghi nhận

SQL Server:

```sql
CREATE ROLE role_backoffice;
GRANT SELECT, INSERT, UPDATE ON dbo.Orders TO role_backoffice;
DENY DELETE ON dbo.Orders TO role_backoffice;
```

MySQL:

```sql
CREATE USER 'ops'@'%' IDENTIFIED BY '...';
GRANT SELECT, INSERT, UPDATE ON hnx_securities.Orders TO 'ops'@'%';
```

Bổ sung: tài khoản dùng chung cho ứng dụng không nên là `sa` / `root`; **ghi nhật ký kiểm toán** (như bảng `AuditLog`) cho mọi thay đổi trạng thái quan trọng.

> **Cảnh báo:** ứng dụng gọi SQL phải dùng **câu lệnh tham số hóa** (parameterized query), không ghép chuỗi từ dữ liệu người dùng — nếu không sẽ bị **SQL injection**, kẻ tấn công có thể chèn thêm `DELETE` / `DROP` vào câu lệnh của bạn.

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
`UPDATE` / `DELETE` không có `WHERE` (hoặc `WHERE` sai)	tác động toàn bảng	viết `SELECT` cùng `WHERE` trước; dùng transaction
`INSERT` không liệt kê cột; số giá trị khác số cột	chèn nhầm cột hoặc hỏng khi bảng đổi cấu trúc	luôn `INSERT INTO t (cột...) VALUES (...)`
Thêm bảng con trước bảng cha; xóa bảng cha trước bảng con	vi phạm khóa ngoại	thêm cha → con; xóa con → cha
Xóa hàng loạt rồi mới nhận ra tiêu chí lọc phụ thuộc dữ liệu vừa xóa (bài 6.3c)	không tính lại được điều kiện	chốt danh sách vào bảng tạm trước
Gán `= NULL` khi so sánh	không dòng nào khớp	`IS NULL` để so sánh, `SET cột = NULL` để gán
Dùng `TRUNCATE` khi còn cần `WHERE`, hoặc bảng có khóa ngoại tham chiếu	mất hết dữ liệu / lỗi	dùng `DELETE ... WHERE`
Quên `COMMIT` hoặc để transaction mở	giữ khóa, chặn người khác	giữ transaction ngắn, luôn kết thúc bằng `COMMIT` / `ROLLBACK`
Tin rằng `ROLLBACK` cứu được mọi thứ	MySQL không hoàn tác DDL; số tự tăng đã cấp không trả lại	không đặt DDL giữa transaction; đừng dùng số tự tăng làm số liên tục
Dùng `GETDATE()` / `NOW()` cho "hôm nay"	kết quả đổi theo ngày, chấm sai	dùng ngày cố định `'2026-09-25'`

Tóm tắt

Chủ đề	Ghi nhớ
INSERT	luôn liệt kê cột; nhiều dòng bằng `VALUES (...), (...)`; `INSERT ... SELECT` để nhân bản; cha trước con
IDENTITY / AUTO_INCREMENT	tự cấp số, không tái sử dụng; `DELETE` không đặt lại, `TRUNCATE` đặt lại
UPDATE	`SET a = ..., b = ...`; luôn `WHERE`; cộng dồn `balance = balance + x`; JOIN khác cú pháp giữa hai hệ
DELETE	từ con lên cha; chốt danh sách trước khi xóa nhiều tầng
DELETE vs TRUNCATE	TRUNCATE: nhanh, không WHERE, reset tự tăng; MySQL không rollback được
Transaction	`BEGIN/START TRANSACTION` → `COMMIT` / `ROLLBACK`; mặc định là autocommit
ACID	Atomicity, Consistency, Isolation, Durability
Best practices	backup, `SELECT` trước, transaction, `@@ROWCOUNT` / `ROW_COUNT()`, quyền tối thiểu

Gợi ý mở rộng

- Viết `TRY...CATCH` (SQL Server) bọc bài 6.4a để tự `ROLLBACK` khi số dư sau khi trừ bị âm.
- Tạo trigger tự ghi `AuditLog` mỗi khi `Customers.status` đổi (Buổi về trigger).
- Thử `SET IDENTITY_INSERT` và quan sát cách cột tự tăng tiếp nối.
- Mở hai cửa sổ Query trong SSMS để quan sát khóa và mức cô lập.
- Thiết kế chính sách xóa mềm cho `Customers` (cột `is_deleted`) và cập nhật các truy vấn báo cáo tương ứng.

Bài tập thực hành của buổi

- **Bài 6.1** — Thêm khách hàng mới (một dòng và nhiều dòng) *(độ khó: dễ)*
- **Bài 6.1** — Thêm lệnh và giao dịch mới cho khách hàng *(độ khó: trung bình)*
- **Bài 6.1** — Nhân bản tài khoản đã đóng vào bảng lưu trữ (INSERT ... SELECT) *(độ khó: trung bình)*
- **Bài 6.1** — Ghi nhật ký với cột tự tăng và giá trị mặc định *(độ khó: trung bình)*
- **Bài 6.2** — Nạp tiền vào tài khoản (UPDATE có WHERE) *(độ khó: dễ)*
- **Bài 6.2** — Chuyển khách hàng mới sang trạng thái kích hoạt *(độ khó: dễ)*
- **Bài 6.2** — Nâng loại khách hàng lên VIP theo tổng tài sản (UPDATE từ subquery) *(độ khó: trung bình)*
- **Bài 6.2** — Cập nhật nhiều cột: chuyển lệnh MP chờ khớp thành LO theo giá đóng cửa *(độ khó: trung bình)*
- **Bài 6.3** — Xóa giao dịch của một khách hàng cụ thể *(độ khó: trung bình)*
- **Bài 6.3** — Xóa lệnh đã hủy cũ (tôn trọng khóa ngoại) *(độ khó: trung bình)*
- **Bài 6.3** — Xóa khách hàng không hoạt động và toàn bộ dữ liệu liên quan *(độ khó: khó)*
- **Bài 6.3** — DELETE và TRUNCATE: dọn bảng nhật ký và đánh số lại *(độ khó: trung bình)*
- **Bài 6.4** — Chuyển tiền giữa hai tài khoản bằng transaction *(độ khó: trung bình)*
- **Bài 6.4** — ROLLBACK và COMMIT: thử xóa lệnh chờ khớp *(độ khó: trung bình)*
- **Bài 6.4** — Khớp lệnh mua trong một transaction (ACID) *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: sửa dữ liệu an toàn và dùng transaction

Buổi này bạn **thay đổi dữ liệu thật**, nên thao tác đúng thói quen quan trọng hơn cú pháp. Cách an toàn nhất trên máy cá nhân: nếu lỡ làm hỏng bảng mẫu, chạy lại file `hnx_securities_sqlserver.sql` để trả mọi thứ về ban đầu (script tự xóa và tạo lại database).

1. Làm việc trên bản sao thay vì bảng gốc.

```
USE hnx_securities;
DROP TABLE IF EXISTS Accounts_thu;
SELECT * INTO Accounts_thu FROM Accounts;      -- bản sao để tập UPDATE/DELETE
```

2. Thói quen "SELECT trước, UPDATE sau". Viết `WHERE` trước, kiểm tra bằng `SELECT`, rồi mới đổi câu lệnh thành `UPDATE`:

```
SELECT account_id, balance FROM Accounts_thu WHERE account_id = 1;       -- xem trước
UPDATE Accounts_thu SET balance = balance + 1000000 WHERE account_id = 1;
```

Sau khi chạy, tab **Messages** báo *(1 row affected)*. Nếu số dòng bị ảnh hưởng khác dự kiến, hãy dừng và kiểm tra.

> **Cảnh báo:** bôi đen thiếu dòng `WHERE` rồi bấm F5 là cách phổ biến nhất để vô tình sửa cả bảng. Nhấn **Ctrl+A** hoặc bỏ bôi đen trước khi chạy.

3. Tập transaction với ROLLBACK.

```
BEGIN TRANSACTION;
UPDATE Accounts_thu SET balance = 0;                -- cố ý sai: không có WHERE
SELECT COUNT(*) AS so_tk_so_du_0 FROM Accounts_thu WHERE balance = 0;   -- 93
SELECT @@TRANCOUNT AS giao_dich_dang_mo;            -- 1
ROLLBACK;                                           -- hoàn nguyên toàn bộ
SELECT COUNT(*) AS so_tk_so_du_0 FROM Accounts_thu WHERE balance = 0;   -- trở lại như ban
        đầu
```

Hãy luôn kết thúc giao dịch bằng `COMMIT` hoặc `ROLLBACK`. Nếu để mở, các dòng bị khóa và cửa sổ khác sẽ **treo**; khi đóng tab, SSMS hỏi *commit uncommitted transactions?*, chọn **No** để hoàn nguyên. Kiểm tra bằng `SELECT @@TRANCOUNT;` (0 là sạch).

4. Sửa nhanh bằng giao diện (chỉ để biết). Chuột phải bảng → **Edit Top 200 Rows** cho phép gõ trực tiếp vào lưới và tự chạy `UPDATE`. Tiện nhưng khó kiểm soát; trong công việc thật nên dùng script có `WHERE` và lưu lại.

5. Sao lưu trước khi làm việc quan trọng. Chuột phải `hnx_securities` → **Tasks** → **Back Up...** → chọn nơi lưu `.bak` → **OK**. Khôi phục: chuột phải **Databases** → **Restore Database...**. (Với database mẫu, chạy lại script nhanh hơn.)

Với MySQL Workbench: nút **Auto-Commit Transactions** trên thanh công cụ bật/tắt tự động commit; dùng `START TRANSACTION; ... ROLLBACK;`. Bật **Safe Updates** (`SET sql_safe_updates = 1;`) để MySQL từ chối `UPDATE` / `DELETE` không có `WHERE` theo khóa.

Tự kiểm tra: làm **bài 6.4** (chuyển tiền bằng transaction) trên bản sao `Accounts_thu`, dùng `ROLLBACK` một lần cho tới khi chắc chắn, rồi nộp lời giải trên Codelab.

Buổi 7 — Mối quan hệ giữa bảng & Normal Form

Về mục lục

Đến buổi này, các bạn đã biết truy vấn từng bảng riêng lẻ. Nhưng dữ liệu của một công ty chứng khoán không nằm trong một bảng: khách hàng có nhiều tài khoản, tài khoản đặt nhiều lệnh, lệnh khớp thành nhiều giao dịch... **Thiết kế cách các bảng liên hệ với nhau** quyết định hệ thống có đáng tin cậy hay không: số dư có bị lệch, một khách hàng có bị "nhân bản" thành hai người, báo cáo cuối ngày có khớp với sổ lệnh hay không.

Buổi này nặng lý thuyết hơn các buổi trước, vì vậy mọi khái niệm đều được gắn với **bộ dữ liệu mẫu của khóa học** (xem mô tả đầy đủ ở Buổi 1, mục *"Bộ dữ liệu mẫu của khóa học"*) và có ví dụ chạy được. Cách học của buổi: **học xong phần nào, làm bài tập phần đó ngay** - khung "Thực hành" sẽ hiện ngay dưới phần lý thuyết tương ứng.

> **Quy ước:** "hôm nay" của bộ dữ liệu là **2026-09-25**. Mọi ví dụ chạy được trên cả SQL Server (T-SQL) và MySQL 8; chỗ nào khác nhau sẽ có tiêu đề nhỏ **SQL Server** / **MySQL**. Ví dụ có nút "Chạy thử" là các câu truy vấn chạy độc lập được trên bộ dữ liệu mẫu.

Mục tiêu buổi học

- Sau buổi này bạn có thể **phân biệt và mô hình hóa** quan hệ 1-1, 1-n, n-n (bảng trung gian) trên nghiệp vụ chứng khoán.
- Sau buổi này bạn có thể **hiểu và khai báo** PRIMARY KEY, FOREIGN KEY, composite key, UNIQUE bằng `CREATE TABLE` / `ALTER TABLE`.
- Sau buổi này bạn có thể **chuẩn hóa** một bảng phẳng qua 1NF, 2NF, 3NF và nói được lợi ích lẫn cái giá phải trả.
- Sau buổi này bạn có thể **đọc và vẽ ER Diagram** (Entity, Attribute, Relationship, cardinality) cho hệ thống chứng khoán.
- Sau buổi này bạn có thể dùng `INFORMATION_SCHEMA` để **liệt kê khóa** và **dò dòng mồ côi** (vi phạm toàn vẹn khóa ngoài).

Bản đồ buổi học

Phần	Nội dung	Thời lượng gợi ý
Mở đầu	Vì sao thiết kế quan hệ quan trọng; ôn bộ dữ liệu mẫu	4 phút
1. Quan hệ giữa bảng	1-1, 1-n, n-n, quan hệ đệ quy (lý thuyết + demo)	10 phút
2. Khóa (lý thuyết)	PK, FK, composite key, UNIQUE, khai báo bằng DDL, `INFORMATION_SCHEMA`	14 phút
Thực hành bài 7.1	Xác định khóa và kiểm tra mô hình (5 bài, gồm chữa bài)	43 phút
2.7 Toàn vẹn khóa ngoài	Dò dòng mồ côi	5 phút
Thực hành bài 7.2	Tìm khóa ngoài, kiểm tra toàn vẹn (4 bài, gồm chữa bài)	30 phút
3. Chuẩn hóa (lý thuyết)	Dị thường, 1NF → 2NF → 3NF, lợi ích/đánh đổi	15 phút
Thực hành bài 7.3	Thiết kế bảng 1-1, n-n, chuẩn hóa về 3NF (3 bài, gồm chữa bài)	31 phút
4. ER Diagram	Thành phần, cardinality, đọc sơ đồ, checklist	7 phút
Vẽ ER + tự kiểm tra	Vẽ trên SSMS/Workbench (không chấm tự động)	15 phút
Tổng kết	Tóm tắt, sai lầm thường gặp, giao bài về nhà	6 phút
Tổng	Lý thuyết + demo ~61 phút, thực hành ~119 phút (khoảng 66%)	**180 phút**

---

1. Khái niệm mối quan hệ (Relationships)

Một **quan hệ** thể hiện rằng các dòng của bảng này "gắn" với các dòng của bảng kia. Trong CSDL quan hệ, sự gắn kết đó được lưu bằng **giá trị**: một cột ở bảng con chứa giá trị khóa chính của bảng cha (đó chính là khóa ngoài, phần 2). Đặc trưng của một quan hệ là **bản số (cardinality)** - mỗi bên có thể có tối đa bao nhiêu dòng.

Loại	Đọc là	Cách hiện thực	Ví dụ chứng khoán
1-1	một - một	FK + UNIQUE (hoặc FK là PK)	Một khách hàng - một người đại diện; một khách hàng - một dòng `CustomerStats`
1-n	một - nhiều	FK đặt ở bảng phía "n"	Một khách hàng - nhiều tài khoản; một lệnh - nhiều giao dịch khớp
n-n	nhiều - nhiều	Bảng trung gian có 2 FK	Nhiều tài khoản - nhiều mã chứng khoán (qua `Portfolio`)

> **Mẹo:** đặt câu hỏi hai chiều. "Một khách hàng có tối đa bao nhiêu tài khoản?" (nhiều) và "Một tài khoản thuộc tối đa bao nhiêu khách hàng?" (một) → quan hệ **1-n**. Nếu cả hai chiều đều trả lời "nhiều" → **n-n**; cả hai đều "một" → **1-1**.

1.1. Quan hệ One-to-One (1-1): Một khách hàng - Một người đại diện

Mỗi khách hàng tổ chức (ngân hàng, quỹ, công ty) đăng ký **đúng một** người đại diện giao dịch; mỗi người đại diện chỉ đại diện cho **một** khách hàng. Cách hiện thực: bảng con có cột khóa ngoài **và cột đó là duy nhất** (UNIQUE), hoặc chính khóa ngoài là khóa chính của bảng con.

```
-- T-SQL (MySQL: đổi NVARCHAR thành VARCHAR)
CREATE TABLE CustomerRep (
  rep_id      INT           NOT NULL PRIMARY KEY,
  customer_id INT           NOT NULL UNIQUE,          -- UNIQUE biến 1-n thành 1-1
  rep_name    NVARCHAR(100) NOT NULL,
  rep_title   NVARCHAR(50)  NULL,
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

Trong bộ dữ liệu mẫu đã có sẵn một quan hệ 1-1 thực thụ: `CustomerStats` có `customer_id` vừa là **khóa chính** vừa là **khóa ngoài** trỏ `Customers`. Mỗi khách hàng có nhiều nhất một dòng thống kê:

```
SELECT s.customer_id, s.trade_count, s.total_trade_value
FROM CustomerStats s WHERE s.customer_id IN (2, 5, 6) ORDER BY 1;
```

Bằng chứng "1-1" bằng số liệu: số dòng bằng số khách hàng khác nhau (không khách nào có hai dòng thống kê).

```
SELECT COUNT(*) AS so_dong, COUNT(DISTINCT customer_id) AS so_khach FROM CustomerStats;
```

Khi nào tách 1-1 thay vì để chung một bảng? (1) tách thông tin ít dùng/nhạy cảm (hồ sơ KYC) ra bảng riêng để phân quyền; (2) tách phần dữ liệu "thống kê" thay đổi liên tục khỏi phần danh tính ổn định; (3) chỉ một phần khách hàng mới có dữ liệu đó (người đại diện chỉ áp dụng cho khách hàng tổ chức).

1.2. Quan hệ One-to-Many (1-n): Một khách hàng - Nhiều tài khoản

Đây là quan hệ phổ biến nhất. Khóa ngoài nằm ở **bảng phía "n"** (bảng con), trỏ về khóa chính của bảng phía "1" (bảng cha). Trong hệ thống của chúng ta có cả một chuỗi 1-n:

```
Customers 1 ---- n Accounts 1 ---- n Orders 1 ---- n Trades
                                        n ---- 1 Securities 1 ---- n Trades
```

Đếm số tài khoản mỗi khách hàng (khách hàng 10 chưa mở tài khoản nào - nhớ "n" có thể bằng 0):

```
SELECT c.customer_id, c.full_name, COUNT(a.account_id) AS so_tk
FROM Customers c LEFT JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.customer_id IN (1, 2, 6, 10, 31) GROUP BY c.customer_id, c.full_name ORDER BY 1;
```

Một lệnh (`Orders`) có thể khớp ra 0, 1 hoặc nhiều giao dịch (`Trades`): lệnh đã hủy/đang chờ chưa có giao dịch nào.

```sql
SELECT o.order_id, o.symbol, o.status, t.trade_id
FROM Orders o LEFT JOIN Trades t ON t.order_id = o.order_id WHERE o.account_id = 2 ORDER BY
        o.order_id;
```

> **Ví dụ nghiệp vụ:** lệnh 105 (PVS) đã `CANCELLED` nên `trade_id` là `NULL`; lệnh 110 (IDC) mới `PARTIAL` - khớp một phần. Khi lệnh khớp thành nhiều đợt, cùng một `order_id` sẽ xuất hiện ở nhiều dòng `Trades`: đó chính là "một lệnh - nhiều giao dịch".

Quan hệ đệ quy (tự tham chiếu). Một bảng có thể là cha của chính nó: `Customers.referrer_id` trỏ về `Customers.customer_id` (khách hàng được ai giới thiệu). Đây vẫn là 1-n (một người giới thiệu - nhiều người được giới thiệu) nhưng cả hai đầu là cùng một bảng. Buổi 8 sẽ dùng *self join* để đọc nó.

1.3. Quan hệ Many-to-Many (n-n): Nhiều khách hàng - Nhiều sản phẩm

Một tài khoản nắm giữ **nhiều** mã; một mã được **nhiều** tài khoản nắm giữ. Không thể đặt khóa ngoài vào bảng nào: nếu đặt `symbol` vào `Accounts`, mỗi tài khoản chỉ có 1 mã; nếu đặt `account_id` vào `Securities`, mỗi mã chỉ thuộc 1 tài khoản. Cách giải duy nhất là **bảng trung gian** (còn gọi bảng giao / junction table / associative table) chứa hai khóa ngoài và tách n-n thành **hai quan hệ 1-n**:

```
Accounts 1 ---- n Portfolio n ---- 1 Securities
                (account_id, symbol)  = khóa chính kép
```

`Portfolio` chính là bảng trung gian, và nó còn mang **thuộc tính của mối quan hệ** (`quantity`, `avg_cost` - thuộc về *cặp* tài khoản-mã chứ không riêng bên nào):

```sql
SELECT p.account_id, p.symbol, p.quantity, p.avg_cost
FROM Portfolio p WHERE p.account_id = 3 ORDER BY p.symbol;
```

Đếm theo chiều ngược lại (mỗi mã có bao nhiêu tài khoản nắm giữ) - đây là "phía n" thứ hai của quan hệ n-n:

```sql
SELECT symbol, COUNT(*) AS so_tai_khoan FROM Portfolio GROUP BY symbol ORDER BY COUNT(*)
        DESC, symbol;
```

Bảng sau tóm tắt vì sao n-n **bắt buộc** cần bảng trung gian:

Cách làm	Kết quả
Nhồi nhiều mã vào một cột (`'VCB,FPT,HPG'`)	Vi phạm 1NF, không JOIN được, không ràng buộc được
Thêm `symbol1, symbol2, symbol3...` vào `Accounts`	Giới hạn số mã, nhiều NULL, truy vấn "ai giữ VCB?" phải xét từng cột
Bảng trung gian `(account_id, symbol)`	Không giới hạn, có khóa chính kép chống trùng, có FK bảo vệ cả hai phía

Bài thực hành 7.3b yêu cầu bạn tự thiết kế bảng trung gian `Watchlist` (khách hàng theo dõi mã) theo đúng mẫu này:

```
-- T-SQL (MySQL: symbol VARCHAR(10))
CREATE TABLE Watchlist (
  customer_id INT           NOT NULL,
  symbol      NVARCHAR(10)  NOT NULL,
  added_date  DATE          NOT NULL,
  PRIMARY KEY (customer_id, symbol),              -- khóa kép: không theo dõi trùng một mã
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY (symbol)      REFERENCES Securities(symbol)
);
```

> **Cảnh báo:** lưu nhiều giá trị vào một ô (`'VCB,FPT,HPG'` trong cột `watch_symbols`) thay vì dùng bảng trung gian là lỗi thiết kế kinh điển - vi phạm 1NF (phần 3), không thể JOIN, không thể ràng buộc.

---

2. Khóa (Key) trong Database

2.1. Primary Key (Khóa chính): Định danh duy nhất

Khóa chính định danh **duy nhất** một dòng trong bảng. Yêu cầu: **không trùng, không NULL, ổn định** (không đổi theo thời gian). Mỗi bảng có **đúng một** khóa chính (có thể gồm nhiều cột).

Hai kiểu chọn khóa:

- **Khóa tự nhiên (natural key):** dữ liệu nghiệp vụ, ví dụ `Securities.symbol` (mã chứng khoán);
- **Khóa đại diện (surrogate key):** số vô nghĩa sinh tự động, ví dụ `Customers.customer_id` , `AuditLog.log_id` . Số thứ tự tự tăng: **SQL Server** `IDENTITY(1,1)` , **MySQL** `AUTO_INCREMENT` .

Vì sao `Customers` dùng `customer_id` làm PK mà `customer_code` chỉ là UNIQUE? Vì mã khách hàng là quy ước nghiệp vụ, có thể bị đổi định dạng; còn `customer_id` không bao giờ đổi và nhỏ gọn khi làm khóa ngoài ở hàng triệu dòng lệnh.

2.2. Foreign Key (Khóa ngoài): Tham chiếu đến bảng khác

Khóa ngoài là cột (hoặc bộ cột) ở bảng con có giá trị **phải tồn tại** trong khóa chính (hoặc khóa UNIQUE) của bảng cha, hoặc là NULL nếu cột cho phép NULL. Nhờ nó CSDL **từ chối** dữ liệu vô lý:

```
INSERT INTO Accounts (account_id, customer_id, account_no, account_type, balance,
        opened_date, status)
VALUES (999, 12345, 'ACC99999', 'CASH', 0, '2026-09-25', 'ACTIVE');   -- khách hàng 12345
        không tồn tại
```

- **SQL Server:** `The INSERT statement conflicted with the FOREIGN KEY constraint ...` (lỗi 547);
- **MySQL:** `ERROR 1452 ... Cannot add or update a child row: a foreign key constraint fails` .

Ngược lại, xóa khách hàng đang có tài khoản cũng bị chặn (`Cannot delete or update a parent row`). Hành vi này có thể tùy biến bằng `ON DELETE` / `ON UPDATE` :

Tùy chọn	Khi xóa dòng cha	Dùng khi
`NO ACTION` / `RESTRICT` (mặc định)	Chặn, báo lỗi	Hầu hết trường hợp nghiệp vụ tài chính
`CASCADE`	Xóa luôn các dòng con	Dữ liệu phụ thuộc hoàn toàn (vd chi tiết tạm)
`SET NULL`	Đặt cột FK của dòng con thành NULL	FK cho phép NULL, con vẫn có ý nghĩa khi mất cha

> **Cảnh báo:** trong nghiệp vụ chứng khoán **không nên `CASCADE` lên lệnh/giao dịch** - dữ liệu giao dịch phải lưu vết vĩnh viễn (và phải giữ cho thanh tra). Thay vào đó dùng cột trạng thái (`status = 'CLOSED'`) để "đóng" thay vì xóa.

> **SQL Server vs MySQL:** khóa ngoài **không tự tạo index** trên cột con ở SQL Server (MySQL/InnoDB thì có). Nên tự tạo index cho các cột FK hay JOIN như `Orders.account_id` (sẽ nói ở Buổi 8).

2.3. Composite Key: Khóa gồm nhiều cột

Khi một cột không đủ định danh dòng, ta ghép nhiều cột. Bộ dữ liệu có hai ví dụ điển hình:

Bảng	Khóa chính kép	Ý nghĩa
`MarketPrices`	`(symbol, trade_date)`	Mỗi mã có đúng một dòng giá mỗi ngày
`Portfolio`	`(account_id, symbol)`	Mỗi tài khoản nắm giữ mỗi mã đúng một dòng

Với khóa kép, **từng cột** vẫn có thể lặp lại, chỉ **tổ hợp** là duy nhất. Vì vậy `Portfolio.account_id` là khóa ngoài **1-n** (một tài khoản, nhiều dòng), không phải 1-1, dù nó nằm trong khóa chính (bài 7.1e).

> **Lưu ý:** thứ tự cột trong khóa kép có ý nghĩa: `PRIMARY KEY (account_id, symbol)` tạo index hỗ trợ tìm theo `account_id` (hoặc cả hai) rất nhanh, nhưng tìm chỉ theo `symbol` thì không tận dụng được index này. Hãy đặt cột hay lọc nhất lên trước.

2.4. Unique Key: Giá trị duy nhất nhưng không là PK

`UNIQUE` cấm trùng lặp nhưng **không** là định danh chính, và một bảng có thể có **nhiều** ràng buộc UNIQUE: `Customers.customer_code` , `Accounts.account_no` .

	PRIMARY KEY	UNIQUE
Số lượng mỗi bảng	Đúng 1	Nhiều
Cho phép NULL	Không	Có
Thường là mục tiêu của FK	Có	Có thể

> **SQL Server vs MySQL (NULL trong UNIQUE):** SQL Server chỉ cho **một** giá trị NULL trong cột UNIQUE (hai NULL coi là trùng); MySQL cho **nhiều** NULL. Vì thế cột `Customers.email` (5 khách chưa có email) **không** đặt UNIQUE được trên SQL Server nếu không dùng thêm *filtered index* `WHERE email IS NOT NULL`.

2.5. Khai báo khóa bằng `CREATE TABLE` và `ALTER TABLE`

Có hai chỗ để khai báo khóa: ngay trong `CREATE TABLE`, hoặc thêm sau bằng `ALTER TABLE ... ADD CONSTRAINT`. Nên **đặt tên ràng buộc** rõ ràng (`PK_`, `FK_`, `UQ_`) để dễ đọc thông báo lỗi và dễ xóa/sửa về sau.

```sql
-- T-SQL (MySQL: NVARCHAR -> VARCHAR)
CREATE TABLE OrderFee (
  order_id   INT           NOT NULL,
  fee_type   NVARCHAR(20)  NOT NULL,
  fee_amount DECIMAL(18,2) NOT NULL,
  CONSTRAINT PK_OrderFee PRIMARY KEY (order_id, fee_type),          -- khóa kép
  CONSTRAINT FK_OrderFee_Orders FOREIGN KEY (order_id)
      REFERENCES Orders(order_id) ON DELETE NO ACTION
);

-- Thêm sau khi bảng đã có: UNIQUE và FOREIGN KEY
ALTER TABLE Customers ADD CONSTRAINT UQ_Customers_code UNIQUE (customer_code);
ALTER TABLE Accounts  ADD CONSTRAINT FK_Accounts_Customers
      FOREIGN KEY (customer_id) REFERENCES Customers(customer_id);
```

Việc	SQL Server	MySQL
Xóa khóa ngoài	`ALTER TABLE Accounts DROP CONSTRAINT FK_Accounts_Customers;`	`ALTER TABLE Accounts DROP FOREIGN KEY FK_Accounts_Customers;`
Xóa UNIQUE	`DROP CONSTRAINT UQ_Customers_code`	`DROP INDEX UQ_Customers_code`
Xóa PRIMARY KEY	`DROP CONSTRAINT PK_OrderFee`	`DROP PRIMARY KEY`
Kiểu chuỗi Unicode	`NVARCHAR(n)`	`VARCHAR(n)` (mặc định utf8mb4)

> **Lưu ý:** thêm khóa ngoài lên bảng **đã có dữ liệu** sẽ thất bại nếu đang tồn tại dòng mồ côi (mục 2.7). Vì vậy quy trình an toàn là: dò mồ côi → sửa/xóa → rồi mới `ADD CONSTRAINT`.

2.6. Xem khóa bằng `INFORMATION_SCHEMA`

Cả SQL Server và MySQL có bộ view chuẩn `INFORMATION_SCHEMA` mô tả cấu trúc CSDL - công cụ quan trọng để **kiểm tra mô hình** bằng câu lệnh (đúng tinh thần các bài 7.1, 7.2):

View	Nội dung
`TABLES`	Danh sách bảng/view (`TABLE_TYPE = 'BASE TABLE'` là bảng thật)
`COLUMNS`	Cột, kiểu dữ liệu, có cho NULL không
`TABLE_CONSTRAINTS`	Ràng buộc: `CONSTRAINT_TYPE` = `PRIMARY KEY` / `UNIQUE` / `FOREIGN KEY` / `CHECK`
`KEY_COLUMN_USAGE`	Cột nào thuộc ràng buộc nào (`ORDINAL_POSITION` = thứ tự trong khóa)
`REFERENTIAL_CONSTRAINTS`	Quan hệ khóa ngoài → ràng buộc được tham chiếu

Liệt kê cột khóa chính của mọi bảng (nối `TABLE_CONSTRAINTS` với `KEY_COLUMN_USAGE`):

SQL Server

```sql
SELECT k.TABLE_NAME, k.COLUMN_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k
  ON k.CONSTRAINT_SCHEMA = tc.CONSTRAINT_SCHEMA AND k.CONSTRAINT_NAME = tc.CONSTRAINT_NAME
       AND k.TABLE_NAME = tc.TABLE_NAME
WHERE tc.CONSTRAINT_TYPE = 'PRIMARY KEY'
ORDER BY LOWER(k.TABLE_NAME), k.ORDINAL_POSITION;
```

MySQL: thêm `AND tc.TABLE_SCHEMA = DATABASE()` vì `INFORMATION_SCHEMA` của MySQL chứa **tất cả** CSDL trên máy chủ (bỏ dòng này sẽ ra dòng trùng lặp); `DATABASE()` là CSDL hiện tại.

```sql
SELECT k.TABLE_NAME, k.COLUMN_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k
  ON k.CONSTRAINT_SCHEMA = tc.CONSTRAINT_SCHEMA AND k.CONSTRAINT_NAME = tc.CONSTRAINT_NAME
       AND k.TABLE_NAME = tc.TABLE_NAME
WHERE tc.CONSTRAINT_TYPE = 'PRIMARY KEY' AND tc.TABLE_SCHEMA = DATABASE()
ORDER BY LOWER(k.TABLE_NAME), k.ORDINAL_POSITION;
```

Kết quả (một phần): `Accounts account_id` , `Customers customer_id` , `MarketPrices symbol` , `MarketPrices trade_date` , `Portfolio account_id` , `Portfolio symbol` … - chú ý `MarketPrices` và `Portfolio` xuất hiện **hai dòng** vì có khóa kép.

Liệt kê khóa ngoài kèm bảng cha - **hai cách khác nhau**:

SQL Server: không có sẵn cột bảng cha trong `KEY_COLUMN_USAGE` ; phải nối qua `REFERENTIAL_CONSTRAINTS.UNIQUE_CONSTRAINT_NAME` :

```
SELECT k.TABLE_NAME, k.COLUMN_NAME, pk.TABLE_NAME AS ref_table, pk.COLUMN_NAME AS ref_column
FROM INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE k  ON k.CONSTRAINT_NAME  = rc.CONSTRAINT_NAME
JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE pk ON pk.CONSTRAINT_NAME =
        rc.UNIQUE_CONSTRAINT_NAME
                                           AND pk.ORDINAL_POSITION = k.ORDINAL_POSITION
ORDER BY LOWER(k.TABLE_NAME), LOWER(k.COLUMN_NAME);
```

MySQL: dùng thẳng `REFERENCED_TABLE_NAME`, `REFERENCED_COLUMN_NAME` trong `KEY_COLUMN_USAGE` (lọc `REFERENCED_TABLE_NAME IS NOT NULL`):

```
SELECT TABLE_NAME, COLUMN_NAME, REFERENCED_TABLE_NAME AS ref_table, REFERENCED_COLUMN_NAME
        AS ref_column
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE TABLE_SCHEMA = DATABASE() AND REFERENCED_TABLE_NAME IS NOT NULL
ORDER BY LOWER(TABLE_NAME), LOWER(COLUMN_NAME);
```

Kết quả rút gọn: `Accounts.customer_id → Customers.customer_id`, `Orders.account_id → Accounts.account_id`, `Trades.order_id → Orders.order_id`, `Customers.referrer_id → Customers.customer_id`...

> **Lưu ý khi chấm bài:** *tên ràng buộc* (`CONSTRAINT_NAME`) do hệ quản trị tự sinh (SQL Server: `PK__Customers__...`, MySQL: `PRIMARY`, `Accounts_ibfk_1`) và **tên schema** cũng khác nhau, nên đừng in chúng ra khi cần kết quả giống nhau trên hai hệ; hãy lọc/JOIN theo `TABLE_NAME`, `COLUMN_NAME`.

> **Mẹo:** trong **SSMS**, cách nhanh nhất để xem khóa của một bảng là mở rộng bảng trong Object Explorer → *Keys* (biểu tượng chìa khóa vàng là PK, xám là FK/UNIQUE) hoặc chạy `EXEC sp_help 'Accounts'`. Trong **MySQL Workbench**: chuột phải bảng → *Alter Table* → tab *Foreign Keys*, hoặc chạy `SHOW CREATE TABLE Accounts;`.

Vừa học xong PK, FK, composite key, UNIQUE và cách đọc chúng từ `INFORMATION_SCHEMA`, hãy làm bài 7.1 để tự xác định khóa và kiểm tra một mô hình dữ liệu.

- **Bài 7.1** — Khóa chính của các bảng *(độ khó: dễ)*
- **Bài 7.1** — Bảng có khóa chính nhiều cột *(độ khó: dễ)*
- **Bài 7.1** — Sơ đồ khóa ngoài của mô hình *(độ khó: trung bình)*
- **Bài 7.1** — Kiểm tra mô hình: PK/UNIQUE/FK từng bảng *(độ khó: trung bình)*
- **Bài 7.1** — Nhận diện quan hệ 1-1 hay 1-n *(độ khó: khó)*

2.7. Kiểm tra toàn vẹn khóa ngoài: tìm dòng "mồ côi"

Dòng mồ côi (orphan) là dòng ở bảng con có giá trị khóa ngoài không tồn tại ở bảng cha. Nếu ràng buộc FK đang bật thì không thể có; nhưng với dữ liệu nhập từ hệ thống cũ, hoặc khi bảng con **chưa từng khai báo FK**, mồ côi hoàn toàn có thể xảy ra (ví dụ tài khoản trỏ tới khách hàng đã bị xóa khỏi danh mục). Truy vấn tiêu chuẩn: `NOT EXISTS`.

```sql
-- số tài khoản mà customer_id không có trong Customers (dữ liệu sạch: 0)
SELECT COUNT(*) AS so_dong_mo_coi
FROM Accounts a
WHERE NOT EXISTS (SELECT 1 FROM Customers c WHERE c.customer_id = a.customer_id);
```

Muốn xem **chi tiết** các dòng mồ côi thay vì đếm, bỏ `COUNT(*)` và `SELECT a.account_id, a.customer_id`.

Với khóa ngoài **cho phép NULL** (như `Customers.referrer_id`), phải loại giá trị NULL, vì NULL nghĩa là "không có cha", không phải mồ côi:

```sql
SELECT COUNT(*) AS so_dong_mo_coi
FROM Customers c
WHERE c.referrer_id IS NOT NULL
  AND NOT EXISTS (SELECT 1 FROM Customers r WHERE r.customer_id = c.referrer_id);
```

> **Cảnh báo:** dùng `NOT IN (SELECT ...)` khi tập con có NULL: `x NOT IN (1, 2, NULL)` cho kết quả UNKNOWN với mọi `x`, khiến truy vấn trả về **rỗng** (thiếu dòng). Ví dụ dưới đây "tìm khách hàng chưa từng giới thiệu ai", hai cách cho hai kết quả khác nhau:

```sql
SELECT
  (SELECT COUNT(*) FROM Customers WHERE customer_id NOT IN (SELECT referrer_id FROM
        Customers)) AS not_in,
  (SELECT COUNT(*) FROM Customers c
    WHERE NOT EXISTS (SELECT 1 FROM Customers r WHERE r.referrer_id = c.customer_id)) AS
        not_exists;
```

`NOT IN` cho **0** (sai, vì `referrer_id` có NULL), `NOT EXISTS` cho số đúng. Dùng `NOT EXISTS` là an toàn hơn. (Ở Buổi 8 ta còn có cách thứ ba: `LEFT JOIN ... WHERE bảng_cha.id IS NULL`.)

Trong bài 7.2c, hệ thống cố ý **tắt kiểm tra khóa ngoài** rồi chèn dữ liệu vi phạm để bạn dò ra. Đó là lệnh mà DBA dùng khi nạp dữ liệu lớn:

	Tắt kiểm tra khóa ngoài	Bật lại
SQL Server	`ALTER TABLE Accounts NOCHECK CONSTRAINT ALL;`	`ALTER TABLE Accounts WITH CHECK CHECK CONSTRAINT ALL;`
MySQL	`SET FOREIGN_KEY_CHECKS = 0;`	`SET FOREIGN_KEY_CHECKS = 1;`

> **Mẹo:** sau khi nạp xong, **luôn chạy truy vấn dò mồ côi** trước khi bật lại ràng buộc. Với SQL Server, `WITH CHECK` buộc kiểm tra cả dữ liệu cũ; thiếu nó, ràng buộc bị đánh dấu "không đáng tin" (*not trusted*) và trình tối ưu bỏ qua.

Vừa học xong cách đọc khóa ngoài và dò dòng mồ côi, hãy làm bài 7.2 để liệt kê khách hàng cùng khóa, tài khoản cùng khóa ngoài và kiểm tra toàn vẹn khóa ngoài bằng chính SQL.

- **Bài 7.2** — Khách hàng cùng mã khách hàng (PK) *(độ khó: dễ)*
- **Bài 7.2** — Tài khoản cùng khóa ngoài trỏ Customers *(độ khó: dễ)*
- **Bài 7.2** — Kiểm tra toàn vẹn khóa ngoài (dòng mồ côi) *(độ khó: trung bình)*
- **Bài 7.2** — Khách hàng chưa mở tài khoản *(độ khó: trung bình)*

---

3. Normalization (Chuẩn hóa)

Chuẩn hóa là quá trình tổ chức lại bảng để **giảm dư thừa** và tránh các **dị thường (anomaly)** khi thêm/sửa/xóa. Ba dạng chuẩn đầu tiên (1NF, 2NF, 3NF) là đủ cho 95% thiết kế thực tế. Mỗi dạng chuẩn "cao hơn" **bao hàm** dạng trước: đã 3NF thì đương nhiên đã 2NF và 1NF.

3.1. Bảng phi chuẩn hóa và ba loại dị thường

Giả sử phòng báo cáo lưu mọi thứ trong **một bảng phẳng** `TradeReport` (dữ liệu lấy từ bộ mẫu):

trade_id	customer_code	customer_name	city	symbol	company_name	sector	quantity	price
32	KH100014	Tran Thi Binh	Da Nang	CEO	Tap doan CEO	Bat dong san	1000	22500
39	KH100014	Tran Thi Binh	Da Nang	SHS	Chung khoan Sai Gon - Ha Noi	Chung khoan	1000	15500
80	KH100014	Tran Thi Binh	Da Nang	IDC	Tong Cong ty IDICO	Ha tang khu cong nghiep	50	51000
40	KH100168	Le Thi Giang	Hai Phong	CEO	Tap doan CEO	Bat dong san	2500	22000

Nhìn bảng ta đã thấy sự lặp: "Tran Thi Binh - Da Nang" lặp 3 lần, "Tap doan CEO - Bat dong san" lặp 2 lần. Sự lặp này sinh ra ba loại **dị thường**:

Dị thường	Tình huống	Hậu quả
Cập nhật	Tran Thi Binh chuyển sang Ha Noi; phải sửa **mọi** dòng của bà	Sửa sót một dòng → dữ liệu mâu thuẫn ("Da Nang" và "Ha Noi" cùng tồn tại)
Thêm	Niêm yết mã mới nhưng chưa có giao dịch nào	Không có chỗ lưu (phải chèn dòng giao dịch "giả")
Xóa	Xóa giao dịch duy nhất của một khách hàng	**Mất luôn** thông tin khách hàng đó

Chuẩn hóa từng bước sẽ loại bỏ các dị thường này. Với mỗi dạng chuẩn dưới đây, hãy đọc theo mẫu **TRƯỚC → SAU**.

3.2. 1NF: Loại bỏ nhóm lặp lại

Điều kiện 1NF: mỗi ô chứa **một giá trị nguyên tử** (không danh sách, không nhóm cột lặp lại như `phone1, phone2, phone3`), mỗi dòng xác định được bằng khóa.

Trường hợp A - danh sách trong một ô.

TRƯỚC (vi phạm 1NF):

customer_id	full_name	phones
1	Nguyen Van An	0910000137; 0988111222

SAU (1NF): tách thành bảng con 1-n `CustomerPhone` với khóa chính kép `(customer_id, phone)`:

customer_id	phone
1	0910000137
1	0988111222

Trường hợp B - nhóm cột lặp lại. Bảng danh mục "ngang" cho mỗi tài khoản:

TRƯỚC (vi phạm 1NF):

account_id	symbol1	qty1	symbol2	qty2	symbol3	qty3
3	CEO	500	NVB	1000	SHS	3000

Tài khoản 3 thực tế giữ **4 mã** (thêm VCB 100 cổ phiếu) - bảng ngang không còn chỗ, phải sửa cấu trúc bảng mỗi khi có tài khoản giữ nhiều mã hơn.

SAU (1NF): mỗi mã một dòng - chính là bảng `Portfolio` của bộ dữ liệu:

account_id	symbol	quantity
3	CEO	500
3	NVB	1000
3	SHS	3000
3	VCB	100

> **Ví dụ nghiệp vụ:** cột `phones = '0910000137; 0988111222'` khiến bộ phận CSKH không thể tìm khách theo số điện thoại bằng `WHERE phone = '0988111222'`; phải dùng `LIKE '%0988111222%'` - vừa chậm (không dùng được index) vừa dễ sai (khớp nhầm một phần số khác).

3.3. 2NF: Loại bỏ phụ thuộc hàm không đầy đủ

Điều kiện 2NF: đã 1NF **và** mọi cột không thuộc khóa phải phụ thuộc vào **toàn bộ** khóa chính (chỉ có ý nghĩa với bảng có khóa kép).

TRƯỚC (vi phạm 2NF): bảng `PortfolioFlat` với khóa `(account_id, symbol)`:

account_id	symbol	quantity	avg_cost	company_name	sector
3	CEO	500	22700	Tap doan CEO	Bat dong san
3	VCB	100	90500	Ngan hang Ngoai thuong	Ngan hang
7	CEO	3000	22700	Tap doan CEO	Bat dong san

- `quantity`, `avg_cost` phụ thuộc vào **cả cặp** (account_id, symbol) → đúng.
- `company_name`, `sector` chỉ phụ thuộc `symbol` (một **phần** của khóa) → **vi phạm 2NF**: "Tap doan CEO" lặp ở mọi tài khoản giữ CEO.

SAU (2NF): tách phần chỉ phụ thuộc `symbol` ra bảng riêng.

Bảng `Portfolio` (chỉ dữ liệu phụ thuộc cả cặp):

account_id	symbol	quantity	avg_cost
3	CEO	500	22700
3	VCB	100	90500
7	CEO	3000	22700

Bảng `Securities` (mỗi mã đúng một dòng):

symbol	company_name	sector
CEO	Tap doan CEO	Bat dong san
VCB	Ngan hang Ngoai thuong	Ngan hang

Đây **chính xác là thiết kế hiện tại của bộ dữ liệu mẫu**. Muốn có lại bảng phẳng như cũ, chỉ cần JOIN hai bảng:

```sql
SELECT p.account_id, p.symbol, p.quantity, p.avg_cost, s.company_name, s.sector
FROM Portfolio p
JOIN Securities s ON s.symbol = p.symbol
WHERE p.account_id = 3
ORDER BY p.symbol;
```

3.4. 3NF: Loại bỏ phụ thuộc bắc cầu

Điều kiện 3NF: đã 2NF **và** không có cột thường phụ thuộc vào cột thường khác (không có phụ thuộc bắc cầu): cột thường chỉ phụ thuộc **khóa, toàn bộ khóa, và không gì ngoài khóa**.

TRƯỚC (vi phạm 3NF): `AccountFlat(account_id PK, customer_id, customer_name, customer_city, balance)` - dữ liệu ba tài khoản của khách Tran Thi Binh:

account_id	customer_id	customer_name	customer_city	balance
2	2	Tran Thi Binh	Da Nang	8000000
3	2	Tran Thi Binh	Da Nang	32000000
4	2	Tran Thi Binh	Da Nang	8600000000

`account_id → customer_id → customer_name, customer_city` : tên và thành phố của khách phụ thuộc `customer_id` (cột thường), chỉ **bắc cầu** phụ thuộc `account_id` . Khách có 3 tài khoản thì tên/thành phố lặp 3 lần.

SAU (3NF): tách khách hàng ra bảng riêng, `Accounts` chỉ giữ `customer_id` làm khóa ngoài.

`Customers` :

customer_id	full_name	city
2	Tran Thi Binh	Da Nang

`Accounts` :

account_id	customer_id	balance
2	2	8000000
3	2	32000000
4	2	8600000000

Đúng thiết kế `Customers` - `Accounts` hiện có. Đổi thành phố của khách chỉ cần sửa **một dòng** duy nhất trong `Customers` .

> **Mẹo nhớ:** cột không thuộc khóa phải phụ thuộc vào *"the key (1NF), the whole key (2NF), and nothing but the key (3NF)"* - tạm dịch: khóa, toàn bộ khóa, và không gì khác ngoài khóa.

Dạng chuẩn	Loại bỏ	Dấu hiệu nhận biết	Cách sửa
1NF	Nhóm lặp, ô không nguyên tử	Danh sách trong ô, `col1, col2, col3`	Tách ra bảng con 1-n
2NF	Phụ thuộc một phần vào khóa kép	Cột chỉ phụ thuộc một nửa khóa	Tách theo phần khóa đó ra bảng riêng
3NF	Phụ thuộc bắc cầu	Cột thường xác định cột thường khác	Tách cột xác định + cột bị xác định ra bảng riêng

3.5. Áp dụng: từ `TradeReport` đến 3NF (bài 7.3c)

Bảng phẳng `TradeReport` ở mục 3.1 tách thành:

- `RptCustomer(customer_code PK, customer_name, city)` - thông tin khách phụ thuộc mã khách;
- `RptSecurity(symbol PK, company_name, sector)` - thông tin mã phụ thuộc mã chứng khoán;
- `RptTrade(trade_id PK, customer_code FK, symbol FK, quantity, price)` - chỉ dữ liệu phụ thuộc trực tiếp giao dịch.

Nạp dữ liệu bằng `INSERT ... SELECT DISTINCT` từ bảng nguồn (bảng cha trước, bảng con sau). Khi cần báo cáo phẳng như cũ, ta **JOIN** ba bảng lại: chuẩn hóa không làm mất khả năng báo cáo, chỉ chuyển gánh nặng sang câu lệnh JOIN (Buổi 8). Ví dụ dưới đây dựng lại đúng bốn dòng của bảng phẳng ở mục 3.1 từ các bảng đã chuẩn hóa của bộ dữ liệu (`Customers` , `Securities` , `Trades` , nối qua `Orders` và `Accounts`):

```
SELECT t.trade_id, c.customer_code, c.full_name, c.city, t.symbol, s.company_name, s.sector,
       t.quantity, t.price
FROM Trades t
JOIN Orders o     ON o.order_id    = t.order_id
JOIN Accounts a   ON a.account_id  = o.account_id
JOIN Customers c  ON c.customer_id = a.customer_id
JOIN Securities s ON s.symbol      = t.symbol
WHERE t.trade_id IN (32, 39, 80, 40)
ORDER BY t.trade_id;
```

3.6. Lợi ích và cái giá

Lợi ích	Cái giá
Không dư thừa, dữ liệu **một nguồn sự thật**	Truy vấn cần JOIN nhiều bảng hơn
Sửa một chỗ, đúng mọi nơi (hết dị thường)	Thiết kế ban đầu phức tạp hơn
Ràng buộc PK/FK bảo vệ tính toàn vẹn	Báo cáo tổng hợp lớn có thể chậm hơn
Bảo trì dễ: thêm thuộc tính vào đúng bảng	Nhiều bảng, cần tài liệu (ER Diagram)

Trong thực tế, đội hệ thống chấp nhận **phi chuẩn hóa có chủ đích** ở những chỗ đọc rất nhiều: `CustomerStats.total_trade_value` chính là dữ liệu **suy ra được** từ `Trades` nhưng được lưu sẵn để dashboard đọc nhanh. Đổi lại phải có cơ chế **giữ đồng bộ** (trigger ở Buổi 9). Quy tắc: *chuẩn hóa trước, phi chuẩn hóa sau và có lý do, có cơ chế đồng bộ.*

> **Cảnh báo:** đừng nhầm "chuẩn hóa" với "chia thật nhiều bảng". Chuẩn hóa quá đà (vd tách riêng bảng "tên đệm", bảng "loại lệnh" chỉ có 2 giá trị) làm truy vấn rối mà không giảm dị thường nào. Dừng ở 3NF, rồi cân nhắc theo nhu cầu đọc/ghi.

Vừa học xong 1NF, 2NF, 3NF và cách thiết kế bảng 1-1, n-n, hãy làm bài 7.3 để tự viết `CREATE TABLE` cho quan hệ 1-1, bảng trung gian n-n và chuẩn hóa một bảng báo cáo phi chuẩn về 3NF (chấm theo trạng thái dữ liệu).

- **Bài 7.3** — Thiết kế quan hệ 1-1: người đại diện *(độ khó: trung bình)*
- **Bài 7.3** — Thiết kế quan hệ n-n: danh mục theo dõi *(độ khó: trung bình)*
- **Bài 7.3** — Chuẩn hóa bảng báo cáo phi chuẩn về 3NF *(độ khó: khó)*

---

4. ER Diagram (Entity-Relationship)

ER Diagram là bản vẽ thiết kế ở mức "khái niệm": cho cả nhóm (lập trình viên, DBA, người nghiệp vụ) nhìn cùng một bức tranh về dữ liệu trước khi viết `CREATE TABLE`. Bản vẽ tốt trả lời được ba câu hỏi: có những **đối tượng** nào, mỗi đối tượng có **thuộc tính** gì, và chúng **liên hệ** với nhau ra sao (bao nhiêu - bao nhiêu).

4.1. Thành phần

Thành phần	Ý nghĩa	Ví dụ
Entity (thực thể)	Đối tượng nghiệp vụ, thường thành một bảng	Customer, Account, Security, Order, Trade
Attribute (thuộc tính)	Đặc điểm của thực thể, thành cột; gạch chân = khóa	full_name, city, **customer_id**
Relationship (mối quan hệ)	Liên hệ giữa thực thể, thành khóa ngoài (hoặc bảng trung gian nếu n-n)	Customer *sở hữu* Account

Mẹo gọi tên: **thực thể là danh từ** (Khách hàng, Tài khoản), **mối quan hệ là động từ** (sở hữu, đặt, khớp thành, nắm giữ).

4.2. Ký hiệu cardinality (Crow's Foot)

Ở đầu mỗi đường nối ghi bản số **tối thiểu** và **tối đa**. Trên hình vẽ chân chim, "một" là **gạch đứng**, "nhiều" là **chân chim** (ba nhánh), "không" là **vòng tròn**:

Viết là	Ý nghĩa	Hình chân chim
`1..1`	đúng một	hai gạch đứng
`0..1`	không hoặc một	vòng tròn và một gạch
`1..n`	một hoặc nhiều	một gạch và chân chim
`0..n`	không hoặc nhiều	vòng tròn và chân chim

Cách đọc: đứng ở thực thể A, nhìn đầu nối phía B để biết "một A có bao nhiêu B".

4.3. ER Diagram của hệ thống chứng khoán

Hình dưới là **sơ đồ quan hệ của toàn bộ bộ dữ liệu mẫu** mà chúng ta dùng suốt khóa học. Mỗi hộp là một thực thể (bảng), mỗi đường nối là một quan hệ khóa ngoài.

Hình minh họa: Sơ đồ ER của bộ dữ liệu mẫu (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: `PK` là khóa chính, `FK` là khóa ngoài, `UQ` là duy nhất. Đường nối có **chân chim ở phía bảng con** (bảng "n") và gạch đứng ở phía bảng cha. Đọc dọc theo chuỗi `Customers → Accounts → Orders → Trades` để thấy quan hệ 1-n nối tiếp; `Portfolio` nằm giữa `Accounts` và `Securities` là bảng trung gian của quan hệ n-n; `CustomerStats` bám vào `Customers` bằng quan hệ 1-1; `Customers.referrer_id` là đường nối vòng về chính bảng (quan hệ đệ quy).

Bảng tra cứu các quan hệ trong hình:

Cha (phía 1)	Con (phía n)	Khóa ngoài (ở bảng con)	Bản số	Loại
`Customers`	`Accounts`	`Accounts.customer_id`	`1..1` - `0..n`	1-n
`Accounts`	`Orders`	`Orders.account_id`	`1..1` - `0..n`	1-n
`Orders`	`Trades`	`Trades.order_id`	`1..1` - `0..n`	1-n
`Securities`	`Orders`, `Trades`, `MarketPrices`	`symbol` ở mỗi bảng	`1..1` - `0..n`	1-n
`Accounts` - `Securities`	`Portfolio`	`Portfolio.account_id`, `Portfolio.symbol`	qua bảng trung gian	n-n
`Customers`	`CustomerStats`	`CustomerStats.customer_id` (cũng là PK)	`1..1` - `0..1`	1-1
`Customers`	`Customers`	`Customers.referrer_id`	`0..1` - `0..n`	đệ quy 1-n

Cách đọc vài quan hệ:

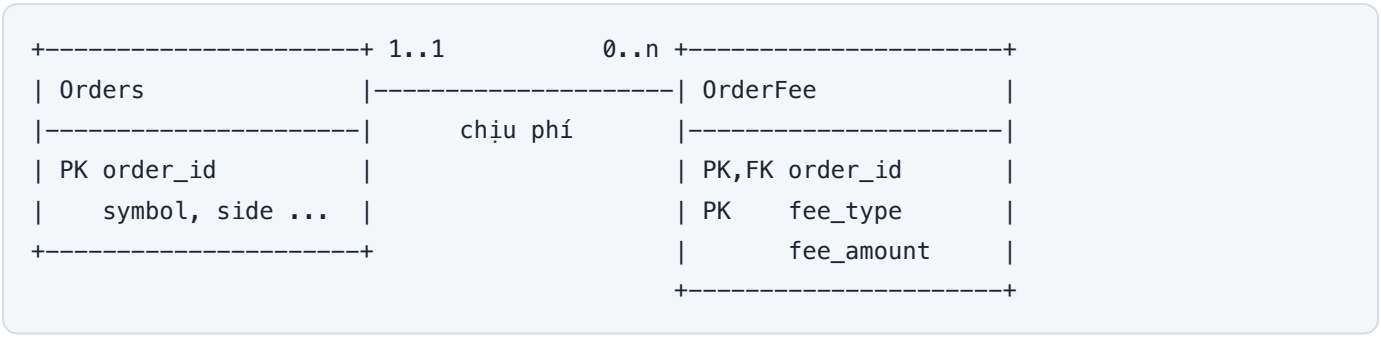
- **Customers 1..1 - 0..n Accounts:** *một* khách hàng có *không hoặc nhiều* tài khoản; *mỗi* tài khoản thuộc *đúng một* khách hàng.
- **Accounts - Securities (n-n)** qua `Portfolio`: `Portfolio` mang khóa chính kép `(account_id, symbol)` gồm hai khóa ngoài.
- **Customers 1..1 - 0..1 CustomerStats:** quan hệ 1-1 tùy chọn về mặt lược đồ (một khách có tối đa một dòng thống kê; trong dữ liệu mẫu cả 40 khách đều có).
- **Customers - Customers (referrer_id):** đệ quy 0..1 người giới thiệu.

> **Lưu ý:** trong hình, bản số "0" ở phía con (`0..n`) cho biết **một khách có thể chưa có tài khoản** - đúng với khách 10 ở mục 1.2. Nếu nghiệp vụ quy định "mở khách hàng là phải mở tài khoản" thì bản số sẽ là `1..n`, và ta phải bảo đảm điều đó bằng ứng dụng hoặc transaction, vì khóa ngoài một mình không ép được "tối thiểu một".

4.4. Vẽ ER Diagram từng bước

Quy trình vẽ một ER cơ bản, áp dụng cho một yêu cầu mới - ví dụ "công ty muốn lưu **phí giao dịch** của mỗi lệnh; một lệnh có thể chịu nhiều loại phí (phí môi giới, phí sàn, thuế TNCN), mỗi loại tối đa một khoản":

1. **Tìm thực thể (danh từ):** `Order` (đã có), `Fee` / khoản phí.
2. **Tìm thuộc tính:** loại phí, số tiền.
3. **Tìm quan hệ và bản số:** một lệnh có `0..n` khoản phí; mỗi khoản phí thuộc `1..1` lệnh → **1-n**.
4. **Chọn khóa:** khoản phí định danh bằng cặp `(order_id, fee_type)` - khóa kép, vì mỗi loại phí xuất hiện tối đa một lần trong một lệnh.
5. **Kiểm tra chuẩn hóa:** không cột nào phụ thuộc một phần khóa hay bắc cầu → đạt 3NF.



Bước 5 chính là mẫu `OrderFee` ở mục 2.5, nên bạn có thể chuyển thẳng từ sơ đồ sang `CREATE TABLE`.

Vẽ bằng công cụ (không chấm tự động):

- **SQL Server / SSMS:** trong Object Explorer, mở CSDL → chuột phải **Database Diagrams** → *New Database Diagram* (lần đầu SSMS hỏi tạo đối tượng hỗ trợ - chọn Yes) → chọn các bảng → *Add*. Các đường nối khóa ngoài tự hiện; chuột phải vào đường nối xem **Properties** để biết bản số; mỗi bảng có biểu tượng chìa khóa ở cột PK.
- **MySQL Workbench:** menu **Database** → **Reverse Engineer...** chọn CSDL → Workbench sinh **EER Diagram** (ký hiệu chân chim). Menu **Model** → **Forward Engineer** làm ngược lại (từ sơ đồ ra script).
- Nếu chưa có công cụ: vẽ tay hoặc dùng bảng/ASCII như trên; điều quan trọng là **đúng khóa và bản số**. Các công cụ web miễn phí như dbdiagram.io hoặc draw.io cũng dùng được.

4.5. Kiểm tra một mô hình đã đúng chưa (checklist)

1. Mỗi bảng có **khóa chính** không? (bài 7.1d: báo cáo `pk_columns = 0` là lỗi);
2. Mọi quan hệ có **khóa ngoài khai báo** không? (so sánh với sơ đồ, bài 7.1c);
3. Quan hệ 1-1 có UNIQUE trên cột FK không? (bài 7.1e);
4. Quan hệ n-n có **bảng trung gian** với khóa chính kép không?;
5. Có bảng nào còn **nhóm lặp/phụ thuộc bắc cầu** không (đạt 3NF)?;
6. Dữ liệu có **dòng mồ côi** không? (mục 2.7).

> **Hoạt động cuối buổi (không chấm tự động, khoảng 15 phút):** (1) dùng SSMS (*Database Diagrams*) hoặc MySQL Workbench (*Reverse Engineer*) để sinh ER Diagram của bộ dữ liệu `hnx_securities`, so khớp với hình ở mục 4.3; (2) thêm bảng `CustomerRep` và `Watchlist` (bài 7.3) vào sơ đồ; (3) tự chấm sơ đồ bằng checklist 4.5. Bài 7.1c, 7.1d, 7.1e là phiên bản chấm được của bước "xác định khóa và kiểm tra mô hình".

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Nhồi nhiều giá trị vào một ô (`'VCB,FPT'`, nhiều số điện thoại)	Vi phạm 1NF, không JOIN/ràng buộc/đánh index được	Tách bảng con hoặc bảng trung gian, mỗi giá trị một dòng
Quan hệ n-n mà không có bảng trung gian	Không giữ được nhiều đối tác, hoặc lặp dữ liệu	Bảng trung gian với hai FK và khóa chính kép
Đặt khóa chính là cột có thể đổi (số điện thoại, họ tên)	Đổi khóa kéo theo sửa hàng loạt bảng con	Dùng khóa đại diện (`IDENTITY` / `AUTO_INCREMENT`) và đặt UNIQUE cho mã nghiệp vụ
Quên khai báo khóa ngoài "vì ứng dụng đã kiểm tra rồi"	Import dữ liệu là xuất hiện dòng mồ côi	Luôn khai báo FK; dò mồ côi trước khi bật lại ràng buộc
Đặt cột tổng hợp (tổng tiền) vào bảng mà không có cơ chế cập nhật	Số liệu tổng lệch so với chi tiết	Tính khi truy vấn, hoặc phi chuẩn hóa có chủ đích kèm trigger/đồng bộ
Dùng `NOT IN (SELECT ...)` khi tập con có NULL	Kết quả rỗng/thiếu dòng mà không báo lỗi	`NOT EXISTS` hoặc `LEFT JOIN ... IS NULL`
`ON DELETE CASCADE` lên lệnh/giao dịch	Xóa một khách hàng làm mất luôn lịch sử giao dịch	Chặn xóa (`NO ACTION`), đóng khách bằng cột `status`
In `CONSTRAINT_NAME` khi so sánh kết quả hai hệ	Kết quả khác nhau vì tên tự sinh	Lọc/nối theo `TABLE_NAME`, `COLUMN_NAME`

Tóm tắt

Chủ đề	Điểm cần nhớ
1-1	Khóa ngoài + UNIQUE (hoặc FK là PK): `CustomerStats`, `CustomerRep`
1-n	Khóa ngoài ở bảng "n": `Customers` - `Accounts` - `Orders` - `Trades`
n-n	Bảng trung gian với hai FK, khóa chính kép: `Portfolio`, `Watchlist`
PK / FK / UQ	PK: duy nhất + không NULL, 1 cái/bảng; FK: trỏ khóa bảng cha; UQ: duy nhất, nhiều cái, NULL tùy hệ
Composite key	Nhiều cột tạo một khóa: `MarketPrices`, `Portfolio`
DDL khóa	`CONSTRAINT tên PRIMARY KEY/FOREIGN KEY/UNIQUE`; xóa FK: `DROP CONSTRAINT` (T-SQL) / `DROP FOREIGN KEY` (MySQL)
1NF / 2NF / 3NF	Ô nguyên tử / phụ thuộc toàn bộ khóa / không phụ thuộc bắc cầu
Lợi ích và cái giá	Ít dư thừa, dễ bảo trì ↔ nhiều JOIN; phi chuẩn hóa có chủ đích phải kèm cơ chế đồng bộ
ER Diagram	Entity - Attribute - Relationship; cardinality `1..1`, `0..1`, `1..n`, `0..n`
`INFORMATION_SCHEMA`	`TABLE_CONSTRAINTS`, `KEY_COLUMN_USAGE`, `REFERENTIAL_CONSTRAINTS`; MySQL nhớ lọc `DATABASE()`
Dòng mồ côi	`NOT EXISTS`; nhớ `IS NOT NULL` với FK cho phép NULL

Bài tập thực hành của buổi

- **Bài 7.1** — Khóa chính của các bảng *(độ khó: dễ)*
- **Bài 7.1** — Bảng có khóa chính nhiều cột *(độ khó: dễ)*
- **Bài 7.1** — Sơ đồ khóa ngoài của mô hình *(độ khó: trung bình)*
- **Bài 7.1** — Kiểm tra mô hình: PK/UNIQUE/FK từng bảng *(độ khó: trung bình)*
- **Bài 7.1** — Nhận diện quan hệ 1-1 hay 1-n *(độ khó: khó)*
- **Bài 7.2** — Khách hàng cùng mã khách hàng (PK) *(độ khó: dễ)*
- **Bài 7.2** — Tài khoản cùng khóa ngoài trỏ Customers *(độ khó: dễ)*
- **Bài 7.2** — Kiểm tra toàn vẹn khóa ngoài (dòng mồ côi) *(độ khó: trung bình)*
- **Bài 7.2** — Khách hàng chưa mở tài khoản *(độ khó: trung bình)*
- **Bài 7.3** — Thiết kế quan hệ 1-1: người đại diện *(độ khó: trung bình)*
- **Bài 7.3** — Thiết kế quan hệ n-n: danh mục theo dõi *(độ khó: trung bình)*
- **Bài 7.3** — Chuẩn hóa bảng báo cáo phi chuẩn về 3NF *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Nội dung syllabus không chấm tự động được: **vẽ ER Diagram** (thao tác đồ họa - làm theo hướng dẫn mục 4.4 và tự đối chiếu bằng checklist 4.5; bài 7.1c/7.1d/7.1e là phiên bản chấm được của bước "xác định khóa và kiểm tra mô hình").

Gợi ý mở rộng

- Thử thiết kế bảng `OrderFees(order_id, fee_type, fee_amount)` với khóa kép và khóa ngoài trỏ `Orders`; tự vẽ ER Diagram mở rộng (mẫu ở mục 2.5 và 4.4).
- Xem `INFORMATION_SCHEMA.COLUMNS` để lập "từ điển dữ liệu": bảng, cột, kiểu, có NULL không.
- Tìm hiểu **BCNF** và **4NF** (những dạng chuẩn cao hơn 3NF) và `CHECK` constraint để ràng buộc miền giá trị (`side IN ('BUY','SELL')`).
- Đọc trước Buổi 8: các quan hệ trên được "nối" lại bằng **JOIN**.

Thực hành trên Windows: khóa, sơ đồ ER và tạo bảng bằng giao diện

Để không làm hỏng dữ liệu mẫu, bạn tạo một database luyện tập riêng.

1. Tạo database luyện tập `hnx_practice`.

```
CREATE DATABASE hnx_practice;
GO
USE hnx_practice;
GO
```

2. Tạo bảng có khóa chính, khóa ngoài, UNIQUE bằng script. Mô hình quan hệ 1-n giữa khách hàng và người phụ trách:

```
CREATE TABLE CustomerRep (
  rep_id    INT           NOT NULL PRIMARY KEY,
  rep_name  NVARCHAR(100) NOT NULL,
  email     VARCHAR(100)  NOT NULL UNIQUE
);
CREATE TABLE Client (
  client_id INT           NOT NULL PRIMARY KEY,
  full_name NVARCHAR(100) NOT NULL,
  rep_id    INT           NULL,
  CONSTRAINT fk_Client_Rep FOREIGN KEY (rep_id) REFERENCES CustomerRep(rep_id)
);
```

3. Thử vi phạm khóa để thấy lỗi.

```
INSERT INTO Client (client_id, full_name, rep_id) VALUES (1, N'Nguyễn Văn An', 99);
-- Lỗi: INSERT statement conflicted with the FOREIGN KEY constraint "fk_Client_Rep"
```

4. Tạo bảng bằng giao diện (Table Designer). Chuột phải **Tables** → **New** → **Table...**: nhập tên cột và kiểu dữ liệu, bỏ tick **Allow Nulls** cho cột bắt buộc, chuột phải ô cột → **Set Primary Key**. Lưu bằng **Ctrl+S**, đặt tên bảng `Watchlist`. Xem lại script tương ứng bằng chuột phải bảng → **Script Table as** → **CREATE To** → **New Query Editor Window**.

5. Vẽ sơ đồ ER cho `hnx_securities`.

1. Mở rộng **hnx_securities**, chuột phải **Database Diagrams** → **New Database Diagram** (lần đầu SSMS hỏi tạo đối tượng hỗ trợ: chọn **Yes**).
2. Chọn tất cả bảng, bấm **Add**, rồi **Close**.
3. Các đường nối khóa ngoài tự hiện, chuột phải vào đường nối → **Properties** để xem bảng cha/con. Chọn **Arrange Selection** hoặc kéo thả để sơ đồ dễ đọc.
4. So khớp với hình ER ở mục 4 của bài giảng, ghi chú các bản số 1-n.

6. Liệt kê toàn bộ khóa ngoài bằng lệnh.

```
USE hnx_securities;
SELECT fk.name AS khoa_ngoai,
       OBJECT_NAME(fk.parent_object_id)     AS bang_con,
       OBJECT_NAME(fk.referenced_object_id) AS bang_cha
FROM sys.foreign_keys fk
ORDER BY bang_con, khoa_ngoai;
```

Với MySQL Workbench: menu **Database** → **Reverse Engineer...** để sinh sơ đồ EER từ database có sẵn; `SHOW CREATE TABLE Accounts;` xem khóa ngoài.

Tự kiểm tra: thêm bảng nối nhiều-nhiều `Watchlist(client_id, symbol)` với khóa chính kép, rồi vẽ lại sơ đồ có cả bảng này.

Buổi 8 — JOIN: Kết hợp bảng

Về mục lục

Buổi 7 đã cho ta bản đồ: khách hàng có tài khoản, tài khoản có lệnh, lệnh có giao dịch. Nhưng bản đồ chỉ để **hiểu**; muốn ra báo cáo thật ("khách hàng nào giao dịch nhiều nhất tuần này", "mã nào không có ai mua", "ai chưa từng giao dịch") ta phải **kết hợp các bảng trong một câu truy vấn**. Công cụ đó là `JOIN`.

Buổi này đi qua các loại JOIN theo thứ tự: INNER, LEFT, RIGHT, FULL, CROSS, Self Join; sau đó áp dụng vào bài toán phân tích (tổng/trung bình giá trị giao dịch, mã bán chạy nhất) và nói nhẹ về hiệu năng (index). Mỗi loại JOIN đều có **ví dụ chạy được ngay** (nút "Chạy thử") và **bài tập ngay bên dưới** - học đến đâu làm đến đó. Mọi ví dụ chạy được trên cả **SQL Server** và **MySQL 8**; chỗ nào cú pháp khác nhau sẽ có tiêu đề nhỏ.

> **Ví dụ nghiệp vụ:** bảng `Trades` chỉ có `order_id` và `symbol`, không có tên khách hàng. Muốn gửi báo cáo khớp lệnh cuối ngày cho từng khách, phòng vận hành phải nối `Trades` → `Orders` → `Accounts` → `Customers`. Không có JOIN thì không có báo cáo nào có ý nghĩa.

> **Quy ước:** bộ dữ liệu mẫu xem mục *"Bộ dữ liệu mẫu của khóa học"* ở Buổi 1 và các quan hệ ở Buổi 7. "Hôm nay" cố định là **2026-09-25**. Trong buổi này, "giao dịch" = `Trades` (kết quả khớp lệnh, có `quantity`, `price`), "sản phẩm" = `Securities` (mã chứng khoán).

Chuỗi khóa ngoài ta sẽ đi qua rất nhiều lần:

```
Customers ──(customer_id)──> Accounts ──(account_id)──> Orders ──(order_id)──> Trades ──(symbol)──> Securities
```

Mục tiêu buổi học

- Sau buổi này bạn có thể **nối bảng bằng INNER JOIN** (2 và nhiều bảng) theo chuỗi khóa ngoài.
- Sau buổi này bạn có thể dùng **LEFT/RIGHT JOIN** để giữ hàng của một phía và **tìm hàng không khớp** (`IS NULL`).
- Sau buổi này bạn có thể giải thích **FULL OUTER JOIN** và viết cách thay thế trên MySQL bằng `UNION`.
- Sau buổi này bạn có thể dùng **CROSS JOIN** đúng chỗ (dựng lưới báo cáo) và biết khi nào không nên dùng.
- Sau buổi này bạn có thể viết **Self Join**: cặp khách cùng thành phố, cây người giới thiệu.
- Sau buổi này bạn có thể **phân tích nghiệp vụ bằng JOIN + GROUP BY** (tổng/trung bình giá trị giao dịch, mã bán chạy) và nhận biết ảnh hưởng của **Index** tới hiệu năng JOIN.

Bản đồ buổi học

Phần	Nội dung	Thời lượng gợi ý
Mở đầu	Tổng quan các loại JOIN (hình Venn), bảng mini	4 phút
1. INNER JOIN	Khái niệm, cú pháp, nhiều bảng, khóa kép	12 phút
Thực hành bài 8.1	INNER JOIN 2 bảng, 3 bảng, 5 bảng (3 bài, gồm chữa bài)	22 phút
1.4 Hiệu năng	Index, kế hoạch thực thi	5 phút
2-5. LEFT / RIGHT / FULL / CROSS	Giữ hàng một phía, anti-join, đối chiếu, lưới báo cáo	22 phút
Thực hành bài 8.2	LEFT, anti-join, RIGHT, FULL, CROSS (5 bài, gồm chữa bài)	48 phút
6. Self Join	Cặp cùng thành phố, cây người giới thiệu	6 phút
Thực hành bài 8.4	Self Join một và hai cấp (2 bài, gồm chữa bài)	22 phút
7. Phân tích bằng JOIN	JOIN + GROUP BY: tổng, trung bình, bán chạy	7 phút
Thực hành bài 8.3	Phân tích giá trị giao dịch (3 bài, gồm chữa bài)	27 phút
Tổng kết	Tóm tắt, sai lầm thường gặp, giao bài về nhà	5 phút
Tổng	Lý thuyết + demo ~61 phút, thực hành ~119 phút (66%)	**180 phút**

Tổng quan các loại JOIN

`JOIN` ghép các dòng của hai bảng thành các dòng rộng hơn, dựa trên một **điều kiện nối** (thường là khóa ngoài = khóa chính). Các loại JOIN chỉ khác nhau ở một điều: **xử lý thế nào với dòng không tìm được đối tác** ở bảng kia. Hình dưới dùng biểu đồ Venn để so sánh năm loại JOIN mà ta sẽ học.

Hình minh họa: Các loại JOIN (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: hai vòng tròn là hai bảng (trái và phải); phần giao nhau là các dòng **khớp** điều kiện; phần tô màu là những dòng **được giữ lại** trong kết quả. INNER chỉ giữ phần giao; LEFT giữ cả vòng trái; RIGHT giữ cả vòng phải; FULL giữ cả hai vòng; riêng CROSS không có điều kiện nối nên ghép mọi dòng với mọi dòng.

Để dễ hình dung, lấy một ví dụ mini: bảng trái là 3 khách hàng, bảng phải là 2 tài khoản (nối theo `customer_id`).

Bảng trái: khách hàng		Bảng phải: tài khoản	
`customer_id` = 1	Nguyen Van An	`ACC00001`	thuộc khách 1
`customer_id` = 10	Ngo Thi Nga	`ACC00002`	thuộc khách 2 (không có trong bảng trái)
`customer_id` = 15	Vo Van Tuan		

Loại JOIN	Kết quả (khách / tài khoản)	Số dòng
INNER	(1, ACC00001)	1
LEFT	(1, ACC00001), (10, NULL), (15, NULL)	3
RIGHT	(1, ACC00001), (NULL, ACC00002)	2
FULL	(1, ACC00001), (10, NULL), (15, NULL), (NULL, ACC00002)	4
CROSS	mọi cặp 3 x 2	6

> **SQL Server vs MySQL:** `INNER`, `LEFT`, `RIGHT`, `CROSS JOIN` chạy giống nhau trên cả hai hệ. Khác biệt lớn nhất: **MySQL không có** `FULL JOIN` (mục 4). Ngoài ra MySQL có `JOIN ... USING (cột)` và `NATURAL JOIN` còn T-SQL không hỗ trợ - khóa học này luôn dùng `JOIN ... ON` để một câu lệnh chạy được trên cả hai.

---

1. INNER JOIN

1.1. Khái niệm

`INNER JOIN` chỉ trả về các cặp dòng **khớp điều kiện ở cả hai bảng**. Dòng nào ở một bên không tìm được dòng tương ứng bên kia thì **biến mất** khỏi kết quả.

1.2. Cú pháp

```
SELECT ...
FROM   bảng1 [AS] b1
INNER JOIN bảng2 [AS] b2 ON b1.cột_khóa = b2.cột_khóa;
```

`INNER` có thể bỏ (`JOIN` mặc định là INNER JOIN). Điều kiện nối đặt sau `ON`, thường là **khóa ngoài = khóa chính**. Dùng **bí danh** (alias) `c`, `a` ... để ngắn gọn và tránh mơ hồ khi hai bảng có cùng tên cột (`customer_id`, `symbol`...).

Ví dụ: khách hàng cùng tài khoản của họ.

```
SELECT c.customer_id, c.full_name, a.account_no
FROM Customers c
INNER JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.customer_id IN (1, 10, 15)
ORDER BY c.customer_id, a.account_no;
```

Khách 10 và 15 **không xuất hiện**: họ chưa có tài khoản nên không có dòng nào để khớp. Đó chính là bản chất của INNER JOIN, và cũng là lý do ta cần LEFT JOIN ở mục 2.

> **Mẹo:** trước khi chạy, hãy **đoán số dòng**. INNER JOIN 1-n cho ra đúng bằng số dòng của bảng "n" có đối tác (ở đây 1 dòng). Tập thói quen này giúp bạn phát hiện ngay khi JOIN sai điều kiện (số dòng bất ngờ tăng vọt hoặc rụng mất).

1.3. Nối nhiều bảng (Multiple INNER JOIN)

Muốn lấy tên khách hàng cho mỗi giao dịch, phải đi hết chuỗi khóa ngoài: **Trades** → **Orders** → **Accounts** → **Customers**. Mỗi `JOIN` thêm một bảng, mỗi lần nối theo một khóa ngoài:

```sql
SELECT t.trade_id, c.full_name, t.symbol, t.quantity, t.price, t.trade_date
FROM Trades t
INNER JOIN Orders    o ON o.order_id    = t.order_id
INNER JOIN Accounts  a ON a.account_id  = o.account_id
INNER JOIN Customers c ON c.customer_id = a.customer_id
WHERE t.trade_date >= '2026-09-24'
ORDER BY t.trade_id;
```

Kết quả có 14 dòng, mở đầu bằng: `2 Quy Dau tu Tang Truong Xanh MBS 2000 25000.00 2026-09-24`, `5 Ngan Hang Nong Nghiep Phat Trien VCB 500 92000.00 2026-09-24` ...

Thêm bảng `Securities` để lấy tên công ty và lọc theo sàn (nối theo `t.symbol`), ta có truy vấn 5 bảng:

```sql
SELECT t.trade_id, c.full_name, s.company_name, t.quantity * t.price AS gross_value
FROM Trades t
INNER JOIN Orders o ON o.order_id = t.order_id
INNER JOIN Accounts a ON a.account_id = o.account_id
INNER JOIN Customers c ON c.customer_id = a.customer_id
INNER JOIN Securities s ON s.symbol = t.symbol
WHERE s.exchange = 'HNX' AND t.trade_date >= '2026-09-21'
ORDER BY t.trade_id;
```

Cách đọc/viết truy vấn nhiều bảng: (1) xác định bảng "gốc" chứa thứ cần đếm/liệt kê (ở đây `Trades`); (2) đi dần ra từng bảng cần thêm thông tin; (3) mỗi JOIN một `ON` khớp đúng khóa; (4) cuối cùng mới viết `WHERE`, `GROUP BY`, `ORDER BY`.

Điều kiện trong `ON` hay `WHERE`? Với INNER JOIN, hai nơi cho kết quả **giống hệt** (`ON` để nối, `WHERE` để lọc - nên tách rõ cho dễ đọc). Với LEFT JOIN thì khác hẳn, xem mục 2.3.

Nối theo khóa kép (nhiều điều kiện trong `ON`). Khi khóa của bảng gồm nhiều cột (`MarketPrices` có khóa `(symbol, trade_date)`), phải nối bằng **đủ các cột** với `AND`. Ví dụ so sánh giá đặt của lệnh với giá đóng cửa của **cùng mã, cùng ngày**:

```sql
SELECT o.order_id, o.symbol, o.order_date, o.price AS gia_dat, m.close_price AS
        gia_dong_cua,
       o.price - m.close_price AS chenh_lech
FROM Orders o
INNER JOIN MarketPrices m ON m.symbol = o.symbol AND m.trade_date = o.order_date
WHERE o.order_id IN (2, 3, 5, 6, 8, 9)
ORDER BY o.order_id;
```

> **Cảnh báo:** thiếu một vế của khóa kép (chỉ `m.symbol = o.symbol`) thì mỗi lệnh sẽ ghép với **mọi ngày** của mã đó - số dòng tăng gấp bội và kết quả sai. Bảng có khóa kép luôn phải nối đủ khóa.

> **Lưu ý:** INNER JOIN này chỉ trả về lệnh nào **có** giá thị trường cùng ngày (72 trên 120 lệnh). Các lệnh còn lại không có dòng giá cùng ngày nên "biến mất" - nếu cần giữ đủ lệnh, dùng LEFT JOIN (mục 2).

Vừa học xong INNER JOIN cho hai, ba và nhiều bảng theo chuỗi khóa ngoài, hãy làm bài 8.1 để nối khách hàng với tài khoản, giao dịch với tên khách hàng và tên sản phẩm.

- **Bài 8.1** — Khách hàng cùng tài khoản của họ (INNER JOIN) *(độ khó: dễ)*
- **Bài 8.1** — Giao dịch cùng tên khách hàng *(độ khó: dễ)*
- **Bài 8.1** — Giao dịch + khách hàng + sản phẩm (nhiều bảng) *(độ khó: trung bình)*

1.4. Hiệu năng: Index và tối ưu hóa

Với dữ liệu mẫu vài trăm dòng, JOIN nào cũng nhanh. Trên hệ thống thật (hàng chục triệu lệnh), tốc độ JOIN phụ thuộc chủ yếu vào việc **cột dùng trong `ON` có index hay không**. Không có index, hệ quản trị phải quét toàn bộ bảng con cho **mỗi** dòng của bảng cha; có index, nó "tra mục lục" và nhảy thẳng tới dòng cần (chi tiết cấu trúc index ở Buổi 10).

- Khóa chính luôn có index. **Khóa ngoài thì không luôn có**: MySQL/InnoDB tự tạo index cho cột FK, còn **SQL Server không** - nên tự tạo cho các cột hay JOIN:

```sql
-- Cú pháp giống nhau ở SQL Server và MySQL
CREATE INDEX IX_Orders_account_id  ON Orders (account_id);
CREATE INDEX IX_Trades_order_id    ON Trades (order_id);
CREATE INDEX IX_Trades_symbol_date ON Trades (symbol, trade_date);   -- index kép cho lọc
        theo mã + ngày
```

- Ba thói quen tối ưu cơ bản: (1) chỉ `SELECT` cột cần dùng, tránh `SELECT *` khi nối nhiều bảng; (2) lọc sớm (`WHERE`) bằng cột có index và **không bọc hàm** quanh cột (`WHERE trade_date >= '2026-09-24'` tốt hơn `WHERE YEAR(trade_date) = 2026`); (3) kiểm tra kế hoạch thực thi.

Việc	SQL Server (SSMS)	MySQL
Xem kế hoạch thực thi	Bấm *Include Actual Execution Plan* (Ctrl+M) rồi chạy	Đặt `EXPLAIN` trước câu SELECT (xem cột `type`, `key`, `rows`)
Xem số trang đọc	`SET STATISTICS IO ON;`	`EXPLAIN ANALYZE` (MySQL 8)
Thuật toán nối thường thấy	Nested Loops / Hash Match / Merge Join	Nested Loop / Hash Join (8.0.18+)

Hai thao tác xem kế hoạch này làm trên máy của bạn, hệ thống chấm bài không kiểm tra.

> **Lưu ý:** đừng thêm index vô tội vạ: mỗi index làm `INSERT/UPDATE` chậm đi và tốn dung lượng. Index phục vụ **truy vấn đọc thường xuyên**.

> **Cảnh báo:** viết kiểu "nối cũ" `FROM Customers c, Accounts a WHERE c.customer_id = a.customer_id`. Nó chạy, nhưng quên mất điều kiện là thành **tích Descartes** (mục 5) trả về hàng chục nghìn dòng vô nghĩa. Luôn dùng `JOIN ... ON`.

---

2. LEFT JOIN (LEFT OUTER JOIN)

2.1. Khái niệm

`LEFT JOIN` trả về **tất cả dòng của bảng trái**, kèm dòng khớp từ bảng phải; nếu không khớp, các cột của bảng phải là `NULL`. Dùng khi câu hỏi có dạng "**tất cả** khách hàng, *kể cả* những người chưa ...".

```sql
SELECT c.customer_id, c.full_name, a.account_no
FROM Customers c
LEFT JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.customer_id IN (1, 10, 15)
ORDER BY c.customer_id, a.account_no;
```

So với INNER JOIN ở mục 1.2, hai khách chưa có tài khoản (10 và 15) đã **xuất hiện** với `NULL`. Cú pháp đầy đủ là `LEFT OUTER JOIN`; từ `OUTER` có thể bỏ.

2.2. Tìm hàng không khớp: `WHERE bảng_phải.id IS NULL`

Sau khi LEFT JOIN, những dòng có cột khóa của bảng phải bằng `NULL` chính là các dòng **không có đối tác**. Đây là kỹ thuật **anti-join** kinh điển: "khách hàng **chưa có** giao dịch nào".

Vì "giao dịch của khách" cách khách hàng 3 bảng (Accounts, Orders, Trades), ta gom "khách hàng đã có giao dịch" thành một bảng dẫn xuất rồi LEFT JOIN vào `Customers`:

```sql
SELECT c.customer_id, c.full_name
FROM Customers c
LEFT JOIN (SELECT DISTINCT a.customer_id
           FROM Accounts a
           INNER JOIN Orders o ON o.account_id = a.account_id
           INNER JOIN Trades t ON t.order_id   = o.order_id) x
       ON x.customer_id = c.customer_id
WHERE x.customer_id IS NULL
ORDER BY c.customer_id;
```

Kết quả có 12 khách; chú ý khách 3 (Le Van Cuong): có tài khoản, có 3 lệnh, nhưng lệnh chưa khớp nên vẫn "chưa có giao dịch". Đây là lý do phải nối tới tận `Trades` chứ không dừng ở `Orders`.

Ba cách viết anti-join:

Cách	Ưu điểm	Lưu ý
`LEFT JOIN ... WHERE phải.khóa IS NULL`	Đọc trực quan, thường tối ưu tốt	Phải chọn cột **khóa** (luôn không NULL khi khớp) để kiểm tra
`NOT EXISTS (SELECT 1 ...)`	An toàn với NULL	Dài hơn một chút
`NOT IN (SELECT ...)`	Ngắn	**Nguy hiểm khi tập con có NULL** (xem Buổi 7)

2.3. Bẫy: điều kiện của bảng phải phải đặt ở `ON`, không phải `WHERE`

So sánh hai truy vấn "tài khoản CLOSED của khách 4, 5, 10":

```sql
-- (A) Điều kiện trong ON: vẫn giữ mọi khách
SELECT c.customer_id, a.account_no, a.status
FROM Customers c LEFT JOIN Accounts a ON a.customer_id = c.customer_id AND a.status =
        'CLOSED'
WHERE c.customer_id IN (4, 5, 10)
ORDER BY c.customer_id, a.account_no;
```

```sql
-- (B) Điều kiện trong WHERE: LEFT JOIN "biến" thành INNER JOIN
SELECT c.customer_id, a.account_no, a.status
FROM Customers c LEFT JOIN Accounts a ON a.customer_id = c.customer_id
WHERE a.status = 'CLOSED' AND c.customer_id IN (4, 5, 10)
ORDER BY c.customer_id, a.account_no;
```

Ở (B), `WHERE a.status = 'CLOSED'` loại luôn các dòng `NULL` nên khách 5 và 10 mất (chỉ còn khách 4). **Quy tắc:** điều kiện *lọc bảng bên phải* của LEFT JOIN phải nằm trong `ON`; điều kiện *lọc bảng bên trái* mới nằm trong `WHERE`. (Bài 8.2c dùng đúng quy tắc này.)

Vị trí điều kiện lọc bảng phải	Ý nghĩa	Kết quả
Trong `ON`	"Chỉ nối những dòng phải thỏa điều kiện"	Giữ đủ dòng bảng trái
Trong `WHERE`	"Chỉ giữ kết quả thỏa điều kiện"	Loại các dòng `NULL` → thành INNER JOIN

2.4. Bẫy: nhân đôi dòng (fan-out) và cách tránh

Khi nối theo chuỗi 1-n-n, dòng của bảng trên bị **lặp** theo số dòng của bảng dưới. Bài 8.2a yêu cầu khách chưa có giao dịch xuất hiện **đúng một dòng** dù họ có nhiều lệnh chưa khớp, vì nếu LEFT JOIN trực tiếp Accounts → Orders → Trades, khách 3 (3 lệnh chưa khớp) sẽ hiện 3 dòng `NULL`. Cách xử lý: nối vào một bảng **đã INNER JOIN sẵn** (như mục 2.2) hoặc `SELECT DISTINCT`.

Nguy hiểm hơn là **cộng sai số**: cộng số dư sau khi nối thêm bảng Orders sẽ lặp số dư theo số lệnh.

```sql
-- Sai: nối thêm Orders, số dư mỗi tài khoản bị cộng lặp theo số lệnh
SELECT c.customer_id, SUM(a.balance) AS sai
FROM Customers c JOIN Accounts a ON a.customer_id = c.customer_id
                 JOIN Orders o   ON o.account_id  = a.account_id
WHERE c.customer_id = 2 GROUP BY c.customer_id;
```

```sql
-- Đúng: chỉ nối Accounts
SELECT c.customer_id, SUM(a.balance) AS dung
FROM Customers c JOIN Accounts a ON a.customer_id = c.customer_id
WHERE c.customer_id = 2 GROUP BY c.customer_id;
```

Nguyên tắc vàng: *tổng hợp ở mức chi tiết nhất trước, rồi mới nối lên* (pre-aggregate trong bảng dẫn xuất). Ta sẽ dùng mẫu này ở mục 7.

> **SQL Server vs MySQL:** truy vấn `COUNT(cột)`, `SUM`, `AVG` trên cột có thể `NULL` (sau LEFT JOIN) đôi khi in thông báo *"Warning: Null value is eliminated by an aggregate or other SET operation"* trên SQL Server. Đó chỉ là cảnh báo, không phải lỗi; kết quả vẫn đúng. Cách tránh: gom nhóm trong bảng dẫn xuất (chỉ chứa dữ liệu thật) rồi LEFT JOIN + `COALESCE`.

---

3. RIGHT JOIN (RIGHT OUTER JOIN)

3.1. Khái niệm và ứng dụng

`RIGHT JOIN` là **ảnh gương** của LEFT JOIN: giữ **tất cả dòng của bảng phải**, bảng trái không khớp thì `NULL`. Hai câu sau **hoàn toàn tương đương**:

```sql
FROM Trades t  RIGHT JOIN Securities s ON s.symbol = t.symbol
FROM Securities s  LEFT JOIN Trades t   ON s.symbol = t.symbol
```

3.2. Khác biệt giữa LEFT và RIGHT

Chỉ khác **thứ tự bảng** trong câu lệnh. Trong thực tế, hầu hết người viết SQL **chỉ dùng LEFT JOIN** (bảng cần giữ đủ dòng đặt đầu tiên, đọc từ trái sang phải cho tự nhiên); RIGHT JOIN hữu ích khi sửa một truy vấn có sẵn mà không muốn đảo cả câu.

	LEFT JOIN	RIGHT JOIN
Bảng được giữ đủ dòng	Bảng **trước** từ khóa JOIN	Bảng **sau** từ khóa JOIN
Dòng không khớp	Cột bảng phải = `NULL`	Cột bảng trái = `NULL`
Viết tương đương	`A LEFT JOIN B`	`B RIGHT JOIN A`

Ví dụ: mỗi mã chứng khoán có bao nhiêu giao dịch **từ 2026-09-22**, mã nào không có thì hiện 0 (bài 8.2c):

```
SELECT s.symbol, COUNT(t.trade_id) AS trade_count, COALESCE(SUM(t.quantity), 0) AS total_qty
FROM Trades t
RIGHT JOIN Securities s ON s.symbol = t.symbol AND t.trade_date >= '2026-09-22'   -- điều
        kiện ngày ở ON!
GROUP BY s.symbol
ORDER BY COUNT(t.trade_id), s.symbol;
```

HPG và PVI có giao dịch trong lịch sử nhưng không có từ ngày 22/09 - nhờ giữ bảng `Securities` nên vẫn hiện với 0. Điều kiện ngày đặt ở `ON` (mục 2.3).

> **Mẹo:** `COUNT(t.trade_id)` đếm theo cột của bảng phía bị `NULL` nên bỏ qua dòng không khớp → ra 0. Nếu dùng `COUNT(*)`, mã không có giao dịch vẫn ra **1** (đếm cả dòng `NULL` được giữ lại). Đây là lỗi rất hay gặp.

---

4. FULL JOIN (FULL OUTER JOIN)

4.1. Khái niệm

`FULL JOIN` trả về **tất cả dòng của cả hai bảng**: khớp thì ghép, không khớp thì phía còn lại là `NULL`. Nó là hợp của LEFT JOIN và RIGHT JOIN. Ứng dụng tiêu biểu là **đối chiếu (reconciliation)** giữa hai nguồn dữ liệu: phát hiện dòng chỉ có ở nguồn này hoặc chỉ có ở nguồn kia.

Bảng tóm tắt các loại JOIN:

JOIN	Giữ dòng khớp	Giữ dòng chỉ có ở bảng trái	Giữ dòng chỉ có ở bảng phải
INNER	Có	Không	Không
LEFT	Có	Có (phải = NULL)	Không
RIGHT	Có	Không	Có (trái = NULL)
FULL	Có	Có	Có

4.2. SQL Server có FULL JOIN, MySQL không

SQL Server:

```
SELECT i.city AS ca_nhan, n.city AS to_chuc
FROM (SELECT DISTINCT city FROM Customers WHERE customer_type = 'INDIVIDUAL') i
FULL JOIN (SELECT DISTINCT city FROM Customers WHERE customer_type = 'INSTITUTION') n ON
        n.city = i.city
ORDER BY COALESCE(i.city, n.city);
```

MySQL (không hỗ trợ `FULL JOIN`): workaround bằng `UNION` của LEFT JOIN và RIGHT JOIN:

```
SELECT i.city AS ca_nhan, n.city AS to_chuc FROM (SELECT DISTINCT city FROM Customers WHERE
        customer_type = 'INDIVIDUAL') i
LEFT JOIN  (SELECT DISTINCT city FROM Customers WHERE customer_type = 'INSTITUTION') n ON
        n.city = i.city
UNION
SELECT i.city, n.city FROM (SELECT DISTINCT city FROM Customers WHERE customer_type =
        'INDIVIDUAL') i
RIGHT JOIN (SELECT DISTINCT city FROM Customers WHERE customer_type = 'INSTITUTION') n ON
        n.city = i.city
ORDER BY 1;
```

`UNION` (không `ALL`) tự loại các dòng khớp bị lặp ở hai nửa. Kết quả ở cả hai hệ:

ca_nhan	to_chuc
Can Tho	NULL
Da Nang	NULL
Ha Noi	Ha Noi
Hai Phong	NULL
TP.HCM	TP.HCM

Thành phố chỉ có khách cá nhân thì cột bên phải `NULL`. Một biến thể chính xác hơn khi sợ trùng dòng là `LEFT JOIN` **`UNION ALL`** phần "chỉ có bên phải": `RIGHT JOIN ... WHERE bảng_trái.id IS NULL` (bài 8.2d dùng cách này).

> **SQL Server vs MySQL:** cùng một yêu cầu "lấy tất cả từ cả hai phía", SQL Server chỉ cần một từ khóa `FULL JOIN`, còn MySQL phải ghép hai truy vấn bằng `UNION`. Khi viết bài chấm chạy được trên cả hai hệ, hãy dùng dạng `UNION` (hoặc `LEFT JOIN` + `UNION ALL` phần chỉ-có-bên-phải) để cùng một logic chạy được ở hai nơi.

4.3. Ví dụ đối chiếu giao dịch với danh mục (bài 8.2d)

Cuối ngày, vận hành so **các cặp (tài khoản, mã) đã giao dịch** với **danh mục đang giữ** (`Portfolio`). Với tài khoản 1 đến 6, kết quả có dạng (rút gọn):

account_id	symbol	trạng thái	Ý nghĩa
2	IDC	`BOTH`	có giao dịch và đang giữ
2	PVS	`BOTH`	có giao dịch và đang giữ
2	SHS	`TRADED_ONLY`	có giao dịch nhưng không còn giữ (đã bán hết?)
1	HPG	`HELD_ONLY`	đang giữ nhưng không có giao dịch trong sổ lệnh (số dư đầu kỳ)

`TRADED_ONLY` / `HELD_ONLY` là các dòng cần điều tra. Nếu chỉ dùng INNER hoặc LEFT JOIN ta chỉ thấy một chiều. Trong bài, dùng `COALESCE(t.account_id, p.account_id)` để lấy khóa từ phía có dữ liệu.

---

5. CROSS JOIN

5.1. Tích Descartes

`CROSS JOIN` ghép **mỗi dòng** của bảng 1 với **mỗi dòng** của bảng 2, không có `ON`. Số dòng = `n1 x n2`. `Customers` (40) x `Securities` (12) cho 480 dòng.

```sql
SELECT s.symbol, s.exchange, ci.city
FROM Securities s
CROSS JOIN (SELECT DISTINCT city FROM Customers WHERE customer_id IN (1, 2)) ci
WHERE s.symbol IN ('FPT', 'VCB')
ORDER BY s.symbol, ci.city;
```

Kết quả 4 dòng (2 mã x 2 thành phố): mỗi mã ghép với cả `Da Nang` và `Ha Noi`.

5.2. Khi nào sử dụng?

CROSS JOIN hiếm khi là "kết quả cuối" mà thường là **bước dựng khung**:

1. **Lưới báo cáo đầy đủ:** tạo mọi tổ hợp (thành phố x loại tài khoản), rồi `LEFT JOIN` số đếm thật để tổ hợp không có dữ liệu hiện `0` thay vì biến mất (bài 8.2e). Cách làm này rất phổ biến trong báo cáo quản trị.
2. **Bảng lịch:** ngày x mã chứng khoán để tìm ngày mã không có giá.
3. Sinh dữ liệu thử, so sánh mọi cặp.

Ví dụ lưới **sàn x chiều lệnh**: đếm số lệnh theo từng tổ hợp (HNX/HOSE x BUY/SELL), kể cả tổ hợp không có lệnh vẫn hiện 0:

```sql
SELECT e.exchange, sd.side, COUNT(o.order_id) AS so_lenh
FROM (SELECT DISTINCT exchange FROM Securities) e
CROSS JOIN (SELECT DISTINCT side FROM Orders) sd
LEFT JOIN Securities s ON s.exchange = e.exchange
LEFT JOIN Orders o     ON o.symbol = s.symbol AND o.side = sd.side
GROUP BY e.exchange, sd.side
ORDER BY e.exchange, sd.side;
```

Mẫu ba bước: **CROSS JOIN dựng khung → LEFT JOIN dữ liệu thật → GROUP BY đếm**. Đây chính là cấu trúc của bài 8.2e.

> **Cảnh báo:** với hai bảng hàng triệu dòng, CROSS JOIN sinh hàng nghìn tỷ dòng và làm treo máy chủ. Luôn kiểm tra kích thước trước (`SELECT COUNT(*)` hai bảng rồi nhân lại). Quên `ON` trong `INNER JOIN` bằng tay ở MySQL cũng vô tình thành CROSS JOIN.

Vừa học xong LEFT, RIGHT, FULL và CROSS JOIN, hãy làm bài 8.2 để giữ đủ dòng bảng trái, tìm khách chưa giao dịch (anti-join), đếm giao dịch theo mã bằng RIGHT JOIN, đối chiếu bằng FULL JOIN và dựng lưới thành phố x loại tài khoản.

- **Bài 8.2** — Tất cả khách hàng kèm giao dịch (LEFT JOIN) *(độ khó: dễ)*
- **Bài 8.2** — Khách hàng chưa có giao dịch nào *(độ khó: trung bình)*
- **Bài 8.2** — Mã chứng khoán và số giao dịch gần đây (RIGHT JOIN) *(độ khó: trung bình)*
- **Bài 8.2** — Đối chiếu giao dịch với danh mục (FULL JOIN) *(độ khó: khó)*
- **Bài 8.2** — Lưới thành phố x loại tài khoản (CROSS JOIN) *(độ khó: trung bình)*

---

6. Self Join

6.1. Kết hợp bảng với chính nó

Self join là JOIN mà cả hai phía là **cùng một bảng**, phân biệt bằng hai **bí danh** khác nhau. Dùng khi một bảng chứa quan hệ **giữa các dòng của chính nó**. Đây không phải một loại JOIN riêng: bạn có thể dùng INNER hay LEFT trong self join, chỉ cần hai bí danh khác nhau.

6.2. Tìm khách hàng ở cùng thành phố

Hai dòng khác nhau, cùng `city`:

```
SELECT c1.city, c1.customer_id AS customer_id_1, c1.full_name AS full_name_1,
       c2.customer_id AS customer_id_2, c2.full_name AS full_name_2
FROM Customers c1
INNER JOIN Customers c2 ON c2.city = c1.city AND c2.customer_id > c1.customer_id
WHERE c1.city = 'Da Nang'
ORDER BY c1.customer_id, c2.customer_id;
```

Kết quả có 10 cặp (5 khách ở Da Nang: 2, 6, 15, 21, 27). Điều kiện `c2.customer_id > c1.customer_id` làm hai việc: **loại việc ghép người với chính mình** và **loại cặp đảo ngược** `(B, A)` khi đã có `(A, B)`. Nếu dùng `<>` thay vì `>`, mỗi cặp bị lặp hai lần.

Điều kiện	Số cặp (5 khách)	Vấn đề
Không có điều kiện thứ hai	25	Có cả (A, A) và cả (A, B), (B, A)
`c2.customer_id <> c1.customer_id`	20	Mỗi cặp lặp hai lần
`c2.customer_id > c1.customer_id`	**10**	Đúng: mỗi cặp một lần

6.3. Cây phân cấp: người giới thiệu (referrer)

`Customers.referrer_id` trỏ tới `customer_id` của người giới thiệu. Để hiển thị **tên** người giới thiệu, nối bảng với chính nó qua `referrer_id`:

```
SELECT c.customer_id, c.full_name, r.full_name AS referrer_name
FROM Customers c
INNER JOIN Customers r ON r.customer_id = c.referrer_id     -- INNER: chỉ khách có người
        giới thiệu
WHERE c.customer_id IN (8, 27, 29, 33)
ORDER BY c.customer_id;
```

Muốn liệt kê cả khách **không** có người giới thiệu thì đổi sang `LEFT JOIN`. Muốn đi **hai cấp** (người giới thiệu của người giới thiệu) thì nối thêm một lần nữa, dùng `LEFT JOIN` vì không phải ai cũng có cấp trên (bài 8.4b):

```
SELECT c.customer_id, c.full_name, r.full_name AS referrer_name, g.full_name AS
        referrer_of_referrer
FROM Customers c
INNER JOIN Customers r ON r.customer_id = c.referrer_id
LEFT  JOIN Customers g ON g.customer_id = r.referrer_id
WHERE c.customer_id IN (8, 27, 29, 33)
ORDER BY c.customer_id;
```

Đọc kết quả: khách 8 (Bui Thi Lan) do Nguyen Van An giới thiệu, người này không có cấp trên (`NULL`); khách 27 (Vu Van Kien) do Bui Thi Lan giới thiệu, và Bui Thi Lan lại do Nguyen Van An giới thiệu - chuỗi **27** ← **8** ← **1**.

> **Lưu ý:** với cây sâu không giới hạn (cấp bao nhiêu cũng được), self join cố định số cấp không đủ; phải dùng **CTE đệ quy** (`WITH ... UNION ALL`), nội dung dành cho khóa nâng cao. Ứng dụng khác của self join: cấu trúc phòng ban - nhân viên - quản lý, so sánh giá ngày hôm nay với hôm qua trong `MarketPrices`.

Vừa học xong Self Join, hãy làm bài 8.4 để tìm cặp khách hàng ở cùng thành phố và dựng chuỗi người giới thiệu hai cấp.

- **Bài 8.4** — Cặp khách hàng ở cùng thành phố (Self Join) *(độ khó: trung bình)*
- **Bài 8.4** — Chuỗi người giới thiệu (Self Join hai cấp) *(độ khó: khó)*

---

7. Phân tích bằng JOIN

Các bài phân tích luôn theo mẫu: **JOIN để đưa dữ liệu về cùng một chỗ → GROUP BY để tổng hợp → ORDER BY/TOP để chọn**.

7.1. Tổng giá trị giao dịch cho từng khách hàng (bài 8.3a)

Giá trị một giao dịch = `quantity * price`. Mẫu chuẩn: **tổng hợp theo khách ở bảng dẫn xuất** (chỉ dữ liệu thật) rồi `LEFT JOIN` từ `Customers` để giữ cả khách chưa giao dịch:

```sql
SELECT c.customer_id, c.full_name,
       COALESCE(x.n, 0) AS trade_count,
       CAST(ROUND(COALESCE(x.v, 0), 2) AS DECIMAL(20,2)) AS total_value
FROM Customers c
LEFT JOIN (SELECT a.customer_id, COUNT(*) AS n, SUM(t.quantity * t.price) AS v
           FROM Accounts a
           INNER JOIN Orders o ON o.account_id = a.account_id
           INNER JOIN Trades t ON t.order_id   = o.order_id
           GROUP BY a.customer_id) x ON x.customer_id = c.customer_id
WHERE c.customer_type = 'INSTITUTION'
ORDER BY total_value DESC, c.customer_id;
```

Đọc kết quả: quỹ Tang Truong Xanh (36) đứng đầu với 4 giao dịch, 422.600.000; hai khách 34 và 40 chưa giao dịch nào nên hiện `0` và `0.00` nhờ `LEFT JOIN` + `COALESCE`.

> **Lưu ý:** `CAST(ROUND(..., 2) AS DECIMAL(20,2))` giúp cả hai hệ in đúng hai chữ số thập phân; T-SQL chia/trung bình số nguyên khác MySQL nên luôn ép kiểu rõ ràng khi cần kết quả giống nhau.

Ví dụ nối nhiều bảng rồi gom nhóm theo một thuộc tính khác (sàn giao dịch), kèm điều kiện lọc trên kết quả tổng hợp bằng `HAVING`:

```sql
SELECT s.exchange, s.sector, COUNT(*) AS trade_count, SUM(t.quantity * t.price) AS
        total_value
FROM Trades t
INNER JOIN Securities s ON s.symbol = t.symbol
GROUP BY s.exchange, s.sector
HAVING COUNT(*) >= 5
ORDER BY total_value DESC, s.exchange, s.sector;
```

7.2. Sản phẩm bán chạy nhất (bài 8.3b)

"Bán chạy" ở đây tính theo **tổng khối lượng khớp của lệnh BÁN**. Cần chiều lệnh (`Orders.side`) và tên công ty (`Securities`):

SQL Server

```sql
SELECT TOP 1 s.symbol, s.company_name, SUM(t.quantity) AS sold_qty
FROM Trades t
JOIN Orders o     ON o.order_id = t.order_id
JOIN Securities s ON s.symbol   = t.symbol
WHERE o.side = 'SELL'
GROUP BY s.symbol, s.company_name
ORDER BY SUM(t.quantity) DESC, s.symbol;
```

MySQL: giữ nguyên câu trên nhưng bỏ `TOP 1` và thêm `LIMIT 1` ở cuối.

```sql
SELECT s.symbol, s.company_name, SUM(t.quantity) AS sold_qty
FROM Trades t
JOIN Orders o     ON o.order_id = t.order_id
JOIN Securities s ON s.symbol   = t.symbol
WHERE o.side = 'SELL'
GROUP BY s.symbol, s.company_name
ORDER BY SUM(t.quantity) DESC, s.symbol
LIMIT 1;
```

Kết quả: `PVS – Tong Cong ty Dich vu Ky thuat Dau khi – 9500`.

> **Cảnh báo:** luôn có **khóa sắp xếp phụ** (`s.symbol`) để khi hai mã đồng hạng kết quả vẫn **xác định**. Thiếu nó, hai hệ (hoặc hai lần chạy) có thể chọn hai mã khác nhau.

7.3. Trung bình giá trị giao dịch của khách hàng (bài 8.3c)

Cùng khuôn mẫu, thay `SUM` bằng `AVG`; khách chưa giao dịch có `AVG = NULL` nên dùng `COALESCE(..., 0)`. Muốn so với **trung bình toàn sàn**, nhúng thêm truy vấn con `(SELECT AVG(quantity * price) FROM Trades)`. Ví dụ đầy đủ cho ba khách 1, 2, 6:

```sql
SELECT c.customer_id,
       CAST(ROUND(COALESCE(x.av, 0), 2) AS DECIMAL(20,2)) AS avg_value,
       CASE WHEN x.av > (SELECT AVG(quantity * price) FROM Trades) THEN 'ABOVE' ELSE 'BELOW'
        END AS vs_market
FROM Customers c
LEFT JOIN (SELECT a.customer_id, AVG(t.quantity * t.price) AS av
           FROM Accounts a
           INNER JOIN Orders o ON o.account_id = a.account_id
           INNER JOIN Trades t ON t.order_id   = o.order_id
           GROUP BY a.customer_id) x ON x.customer_id = c.customer_id
WHERE c.customer_id IN (1, 2, 6)
ORDER BY c.customer_id;
```

Đọc kết quả: khách 6 trung bình `50083333.33` → `ABOVE`; khách 2 `12950000.00` → `BELOW`; khách 1 chưa giao dịch nên `0.00` → `BELOW` (so sánh `NULL > số` cho `UNKNOWN`, rơi vào nhánh `ELSE`).

> **Ví dụ nghiệp vụ:** ban quản lý muốn phân khúc khách hàng theo "quy mô giao dịch trung bình" để chăm sóc khác nhau: khách `ABOVE` được phân cho chuyên viên cao cấp. Cột `vs_market` chính là nhãn phân khúc đơn giản nhất.

Sai lầm thường gặp: (1) `AVG` trên cột đã nhân đôi dòng do JOIN thừa bảng (mục 2.4); (2) quên rằng `AVG` bỏ qua NULL nên khách chưa giao dịch bị loại chứ không phải bằng 0; (3) so sánh chuỗi trung bình đã làm tròn thay vì giá trị gốc; (4) quên `GROUP BY` đủ các cột không nằm trong hàm tổng hợp.

Vừa học xong cách tổng hợp bằng JOIN + GROUP BY, hãy làm bài 8.3 để tính tổng giá trị giao dịch từng khách, tìm mã bán chạy nhất và so sánh trung bình giá trị giao dịch với thị trường.

- **Bài 8.3** — Tổng giá trị giao dịch từng khách hàng *(độ khó: trung bình)*
- **Bài 8.3** — Mã chứng khoán bán chạy nhất *(độ khó: trung bình)*
- **Bài 8.3** — Trung bình giá trị giao dịch từng khách hàng *(độ khó: trung bình)*

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Quên `ON` hoặc viết kiểu nối cũ `FROM a, b WHERE ...` thiếu điều kiện	Tích Descartes: hàng nghìn dòng vô nghĩa	Luôn dùng `JOIN ... ON`
Đặt điều kiện lọc bảng phải của LEFT JOIN trong `WHERE`	LEFT JOIN "biến" thành INNER JOIN, mất khách chưa có dữ liệu	Đặt điều kiện bảng phải trong `ON`
Nối thiếu một vế của khóa kép	Mỗi dòng ghép với nhiều dòng, số dòng phình ra	Nối đủ mọi cột của khóa
Cộng/đếm sau khi nối bảng "n" thứ hai (fan-out)	Tổng bị nhân lên, số liệu sai mà không báo lỗi	Tổng hợp ở bảng dẫn xuất trước, rồi mới nối
Dùng `COUNT(*)` sau LEFT/RIGHT JOIN để đếm "số giao dịch"	Khách/mã không có giao dịch vẫn ra 1	`COUNT(cột_bảng_phải)`
Tìm "chưa có" bằng `NOT IN` khi cột có NULL	Kết quả rỗng/thiếu dòng	`LEFT JOIN ... IS NULL` hoặc `NOT EXISTS`
Self join dùng `<>` thay vì `>` để bỏ cặp lặp	Mỗi cặp xuất hiện hai lần	`c2.id > c1.id`
`FULL JOIN` trên MySQL	Lỗi cú pháp	`LEFT JOIN` `UNION` `RIGHT JOIN`
Sắp xếp Top-N không có khóa phụ	Kết quả không tất định khi đồng hạng	Thêm khóa phụ (`symbol` , `id`)

Tóm tắt

Chủ đề	Điểm cần nhớ
Tổng quan	Các loại JOIN khác nhau ở cách xử lý dòng không khớp (hình Venn)
INNER JOIN	Chỉ giữ dòng khớp cả hai bảng; nhiều bảng: nối lần lượt theo chuỗi khóa ngoài; khóa kép nối đủ cột
LEFT JOIN	Giữ mọi dòng bảng trái; không khớp → `NULL` ; điều kiện bảng phải đặt ở `ON`
Anti-join	`LEFT JOIN ... WHERE bảng_phải.khóa IS NULL` = "chưa có ..."
RIGHT JOIN	Ảnh gương của LEFT JOIN (đổi thứ tự bảng); `COUNT(cột)` không `COUNT(*)`
FULL JOIN	T-SQL: `FULL JOIN` ; MySQL: `LEFT JOIN` `UNION` `RIGHT JOIN` (hoặc `UNION ALL` + `IS NULL`)
CROSS JOIN	Tích Descartes `n1 x n2` ; dựng lưới báo cáo rồi LEFT JOIN số đếm
Self Join	Hai bí danh của cùng một bảng; `c2.id > c1.id` để bỏ cặp lặp; cây phân cấp qua `referrer_id`
Phân tích	Tổng hợp trước (bảng dẫn xuất) rồi nối; `COALESCE` ; sắp xếp có khóa phụ
Hiệu năng	Index cho cột `ON` /FK; không bọc hàm quanh cột lọc; xem kế hoạch thực thi

Bài tập thực hành của buổi

- **Bài 8.1** — Khách hàng cùng tài khoản của họ (INNER JOIN) *(độ khó: dễ)*
- **Bài 8.1** — Giao dịch cùng tên khách hàng *(độ khó: dễ)*
- **Bài 8.1** — Giao dịch + khách hàng + sản phẩm (nhiều bảng) *(độ khó: trung bình)*
- **Bài 8.2** — Tất cả khách hàng kèm giao dịch (LEFT JOIN) *(độ khó: dễ)*
- **Bài 8.2** — Khách hàng chưa có giao dịch nào *(độ khó: trung bình)*
- **Bài 8.2** — Mã chứng khoán và số giao dịch gần đây (RIGHT JOIN) *(độ khó: trung bình)*
- **Bài 8.2** — Đối chiếu giao dịch với danh mục (FULL JOIN) *(độ khó: khó)*
- **Bài 8.2** — Lưới thành phố x loại tài khoản (CROSS JOIN) *(độ khó: trung bình)*
- **Bài 8.3** — Tổng giá trị giao dịch từng khách hàng *(độ khó: trung bình)*
- **Bài 8.3** — Mã chứng khoán bán chạy nhất *(độ khó: trung bình)*
- **Bài 8.3** — Trung bình giá trị giao dịch từng khách hàng *(độ khó: trung bình)*
- **Bài 8.4** — Cặp khách hàng ở cùng thành phố (Self Join) *(độ khó: trung bình)*
- **Bài 8.4** — Chuỗi người giới thiệu (Self Join hai cấp) *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Nội dung không chấm tự động được: **kế hoạch thực thi và hiệu năng Index** (xem mục 1.4, tự thử `CREATE INDEX` và so sánh Execution Plan / `EXPLAIN` trên máy của bạn).

Gợi ý mở rộng

- Đếm số khách hàng được mỗi người giới thiệu (`GROUP BY r.customer_id`) và tìm "người giới thiệu xuất sắc nhất".
- Viết lại các truy vấn `NOT EXISTS` ở Buổi 7 thành `LEFT JOIN ... IS NULL` và so sánh kế hoạch thực thi.

- Tìm hiểu `CROSS APPLY` / `OUTER APPLY` (SQL Server) và `LATERAL` (MySQL 8) - "JOIN với truy vấn con phụ thuộc dòng".
- Chuẩn bị Buổi 9: các thao tác tính tổng giá trị giao dịch như 8.3a sẽ được **tự động cập nhật vào `CustomerStats` bằng trigger**.

Thực hành trên Windows: dựng JOIN bằng giao diện và đọc kế hoạch thực thi

1. Dựng JOIN bằng Query Designer. Trong cửa sổ truy vấn trống, chọn menu **Query → Design Query in Editor...** (hoặc **Ctrl+Shift+Q**):

1. Hộp **Add Table** hiện ra: chọn `Customers` và `Accounts`, bấm **Add**, rồi **Close**.
2. SSMS tự nối hai bảng theo khóa ngoài (đường nối giữa `customer_id` hai bên). Tick các cột muốn lấy: `full_name`, `account_no`, `balance`.
3. Xem câu lệnh sinh ở khung dưới, bấm **OK** để chép vào cửa sổ truy vấn, chạy bằng **F5**.
4. Nhấp đôi vào biểu tượng giữa đường nối để đổi kiểu JOIN (*Select all rows from Customers* tương đương `LEFT JOIN`).

Query Designer giúp bạn hình dung, nhưng câu lệnh sinh ra thường dài; hãy tập viết tay lại gọn gàng và đặt bí danh bảng (`c`, `a`).

2. Kiểm tra số dòng trước và sau JOIN. Một lỗi hay gặp: JOIN làm nhân dòng. Hãy đối chiếu:

```sql
SELECT COUNT(*) AS so_khach_hang FROM Customers;
SELECT COUNT(*) AS so_dong_join
FROM Customers c
INNER JOIN Accounts a ON a.customer_id = c.customer_id;
SELECT COUNT(*) AS so_dong_left_join
FROM Customers c
LEFT JOIN Accounts a ON a.customer_id = c.customer_id;
```

`LEFT JOIN` không bao giờ ít dòng hơn bảng bên trái; chênh lệch giữa `INNER` và `LEFT` chính là số khách hàng chưa có tài khoản.

3. Xem kế hoạch thực thi của một JOIN.

1. Nhấn **Ctrl+M** (*Include Actual Execution Plan*).
2. Chạy:

```sql
SELECT c.full_name, SUM(a.balance) AS tong_so_du
FROM Customers c
INNER JOIN Accounts a ON a.customer_id = c.customer_id
GROUP BY c.full_name
ORDER BY tong_so_du DESC;
```

3. Mở tab **Execution Plan**, đọc từ **phải sang trái**. Tìm các hộp **Nested Loops**, **Hash Match** hoặc **Merge Join**: đó là cách SQL Server thực hiện JOIN. Di chuột vào mũi tên để xem số dòng thực tế.

4. Tắt kế hoạch bằng **Ctrl+M** lần nữa khi không cần.

4. Self Join (khách hàng được giới thiệu). Chạy thử rồi so với hình sơ đồ ở Buổi 7:

```
SELECT kh.customer_id, kh.full_name, gt.full_name AS nguoi_gioi_thieu
FROM Customers kh
LEFT JOIN Customers gt ON gt.customer_id = kh.referrer_id
WHERE kh.referrer_id IS NOT NULL
ORDER BY kh.customer_id;
```

Với MySQL Workbench: chuột phải kết quả hoặc dùng nút **Visual Explain** trên thanh công cụ để xem kế hoạch; `FULL OUTER JOIN` không có sẵn, thay bằng `UNION` của hai `LEFT JOIN`.

Tự kiểm tra: dựng lại **bài 8.2** (khách hàng chưa có giao dịch) bằng Query Designer, tự viết lại gọn, rồi nộp trên Codelab.

Buổi 9 — Stored Procedure & Trigger (cơ bản)

Về mục lục

Đến giờ chúng ta mới chỉ **gõ từng câu lệnh** rồi bấm chạy. Ở công ty chứng khoán, rất nhiều thao tác lặp đi lặp lại mỗi ngày: tính tổng tài sản khách hàng, khóa tài khoản ngủ đông, cập nhật số liệu thống kê sau mỗi phiên khớp lệnh, ghi vết mỗi lần đổi trạng thái khách hàng để phục vụ kiểm toán... Buổi này học cách **đóng gói** các câu lệnh thành **Stored Procedure** (gọi khi cần) và **Trigger** (tự chạy khi có sự kiện), để mọi người trong phòng dùng chung một quy trình chuẩn thay vì mỗi người tự viết một kiểu.

> **Lưu ý:** Buổi này dùng bộ dữ liệu **hnx_securities** đã giới thiệu ở Buổi 1 (xem lại sơ đồ bảng ở mục "Bộ dữ liệu mẫu của khóa học"). Hai bảng **CustomerStats** (thống kê giao dịch theo khách hàng) và **AuditLog** (nhật ký đổi trạng thái khách hàng) được thiết kế riêng cho buổi hôm nay. Ngày "hôm nay" của bộ dữ liệu là cố định: **2026-09-25**.

Mục tiêu buổi học

Sau buổi này bạn có thể:

- Giải thích Stored Procedure là gì và nêu 4 lợi ích: tái sử dụng, bảo mật, hiệu năng, kiểm soát.
- Viết procedure có tham số INPUT/OUTPUT, biến cục bộ (`DECLARE`), điều khiển luồng `IF` / `CASE` / `WHILE` , trả kết quả bằng `SELECT` , `OUTPUT` và `RETURN` trên SQL Server và MySQL.
- Gọi procedure (`EXEC` / `CALL`), xử lý kết quả trả về và viết procedure thay đổi dữ liệu (`UPDATE`) an toàn với `@@ROWCOUNT` / `ROW_COUNT()` .
- Phân biệt trigger `BEFORE` / `AFTER` / `INSTEAD OF` , dùng `NEW` / `OLD` (MySQL) và `inserted` / `deleted` (SQL Server, xử lý theo tập dòng).
- Xây dựng trigger cập nhật số liệu thống kê (`CustomerStats`) và ghi nhật ký kiểm toán (`AuditLog`).
- Quyết định khi nào nên đặt logic trong Stored Procedure/Trigger, khi nào để ở code ứng dụng; biết cách quản lý version, kiểm thử, tài liệu hóa.

Bản đồ buổi học

Buổi học 180 phút, khoảng 34% lý thuyết + demo và 66% thực hành. Mỗi phần lý thuyết đều kết thúc bằng một khung **Thực hành** ngay tại chỗ: học đến đâu làm đến đó.

Phần	Nội dung	Hình thức	Thời lượng gợi ý
1	Stored Procedure là gì? Lợi ích, cú pháp cơ bản	Lý thuyết + demo	8 phút
2	Viết Stored Procedure: tham số, biến, `IF` / `CASE` / `WHILE`, `RETURN` và `SELECT`	Lý thuyết + demo	16 phút
	Thực hành bài 9.1 (5 bài, gồm chữa bài)	Thực hành	43 phút
3	Gọi procedure, xử lý kết quả trả về, procedure có `UPDATE`	Lý thuyết + demo	6 phút
	Thực hành bài 9.2 (3 bài, gồm chữa bài)	Thực hành	31 phút
4	Trigger là gì? Sự kiện, thời điểm, `NEW` / `OLD` và `inserted` / `deleted`	Lý thuyết + demo	8 phút
5	Viết Trigger: AFTER INSERT / UPDATE / DELETE, BEFORE, INSTEAD OF	Lý thuyết + demo	12 phút
	Thực hành bài 9.3 (4 bài, gồm chữa bài)	Thực hành	45 phút
6	Best Practices: khi nào dùng SP/Trigger, version, test, tài liệu	Thảo luận	11 phút
	Tổng	Lý thuyết + demo ~61 phút, thực hành ~119 phút (66%)	**180 phút**

---

1. Stored Procedure là gì?

Stored Procedure (SP, thủ tục lưu trữ) là một tập hợp các câu lệnh SQL được **đặt tên và lưu ngay trong cơ sở dữ liệu**. Muốn chạy, ta chỉ cần gọi tên (kèm tham số nếu có), thay vì gõ lại toàn bộ câu lệnh.

> **Ví dụ nghiệp vụ:** thay vì mỗi cuối tháng bạn tự viết 15 dòng SQL để "tăng lãi cho khách VIP", bạn viết **một lần**, đặt tên `sp_BoostVipBalance`, rồi mọi người trong phòng chỉ cần `EXEC sp_BoostVipBalance 2.5`.

1.1. Lợi ích

Lợi ích	Giải thích	Ví dụ trong công ty chứng khoán
Tái sử dụng	Viết một lần, gọi ở nhiều nơi (báo cáo, ứng dụng, job đêm)	Cùng một SP tính tổng tài sản cho báo cáo tuần và màn hình tra cứu
Bảo mật	Cấp quyền `EXECUTE` trên SP mà **không** cấp quyền đọc/ghi trực tiếp lên bảng	Nhân viên chăm sóc khách hàng gọi được SP đổi trạng thái nhưng không `UPDATE` tự do bảng Customers
Hiệu năng	Server phân tích (biên dịch) sẵn, dùng lại kế hoạch thực thi; giảm lưu lượng mạng vì chỉ gửi tên SP + tham số	Ứng dụng gửi `EXEC sp_TopCustomers 10` thay vì một câu SQL dài
Kiểm soát / nhất quán	Quy tắc nghiệp vụ nằm một chỗ; sửa một nơi là mọi nơi cùng đổi	Quy tắc "VIP = giao dịch từ 200 triệu" nằm trong SP, không rải rác trong nhiều ứng dụng

1.2. Cú pháp cơ bản

Hai hệ quản trị của khóa học dùng cú pháp khác nhau. **SQL Server (T-SQL) là công cụ chính**, MySQL là phương án thứ hai.

SQL Server

```
CREATE PROCEDURE ten_procedure
    @tham_so1 KIEU_DU_LIEU,
    @tham_so2 KIEU_DU_LIEU = gia_tri_mac_dinh,
    @ket_qua  KIEU_DU_LIEU OUTPUT
AS
BEGIN
    SET NOCOUNT ON;      -- không in "n rows affected", gọn kết quả trả về
    -- các câu lệnh SQL
END
```

MySQL

```
DELIMITER //
CREATE PROCEDURE ten_procedure(
    IN  p_tham_so1 KIEU_DU_LIEU,
    OUT p_ket_qua  KIEU_DU_LIEU
)
BEGIN
    -- các câu lệnh SQL, mỗi câu kết thúc bằng dấu ;
END //
DELIMITER ;
```

Vì sao MySQL cần `DELIMITER`? Trong thân procedure có nhiều dấu `;`, nhưng client hiểu `;` là "kết thúc câu lệnh, gửi đi ngay". Lệnh `DELIMITER //` báo cho client tạm coi `//` mới là dấu kết thúc, để cả khối `CREATE`

`PROCEDURE ... END` được gửi đi nguyên vẹn; cuối cùng trả lại `DELIMITER ;`.

> **Cảnh báo:** với SQL Server trên Codelab, script có `CREATE PROCEDURE` / `CREATE TRIGGER` / `CREATE FUNCTION` / `CREATE VIEW` phải **bắt đầu bằng một dòng `GO`**, và các khối cách nhau bằng `GO`. Lý do: `CREATE ...` bắt buộc là câu **đầu tiên của batch**, mà hệ thống chấm tự chèn `SET NOCOUNT ON;` vào trước script của bạn. `GO` không phải T-SQL mà là lệnh của công cụ (SSMS, sqlcmd) báo "kết thúc một batch, gửi đi". Quên `GO` là lỗi số một khi làm bài của buổi này.

SQL Server

```
GO
CREATE PROCEDURE sp_Hello AS
BEGIN
    SELECT 'Xin chao' AS loi_chao;
END
GO
EXEC sp_Hello;
```

MySQL

```
DELIMITER //
CREATE PROCEDURE sp_Hello()
BEGIN
    SELECT 'Xin chao' AS loi_chao;
END //
DELIMITER ;
CALL sp_Hello();
```

Cả hai cho một dòng kết quả `Xin chao`. Bấm **Chạy thử** để tự kiểm tra.

1.3. Sửa, xóa và liệt kê procedure

Việc	SQL Server	MySQL
Tạo mới hoặc thay thế	`CREATE OR ALTER PROCEDURE ...` (SQL Server 2016 SP1+) hoặc `ALTER PROCEDURE ...`	`DROP PROCEDURE IF EXISTS p;` rồi `CREATE PROCEDURE p ...`
Xóa	`DROP PROCEDURE sp_Hello;`	`DROP PROCEDURE sp_Hello;`
Liệt kê	`SELECT name FROM sys.procedures;`	`SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_SCHEMA = DATABASE();`

> **Mẹo:** tránh tiền tố `sp_` trong dự án thật trên SQL Server (SQL Server tìm SP `sp_...` trong database `master` trước nên hơi chậm và dễ trùng với SP hệ thống); nhiều nhóm dùng `usp_` (user stored procedure) hoặc `proc_`. Trong khóa học ta vẫn dùng `sp_` cho dễ nhận biết và đúng với tên bài tập.

---

2. Viết Stored Procedure

2.1. Procedure đầu tiên: một tham số INPUT

Bài toán: lấy khách hàng theo thành phố.

SQL Server

```
GO
CREATE PROCEDURE sp_CustomersByCity @city NVARCHAR(30)
AS
BEGIN
    SET NOCOUNT ON;
    SELECT customer_id, full_name, status
    FROM Customers
    WHERE city = @city
    ORDER BY customer_id;
END
GO
EXEC sp_CustomersByCity 'Da Nang';
```

MySQL

```
DELIMITER //
CREATE PROCEDURE sp_CustomersByCity(IN p_city VARCHAR(30))
BEGIN
    SELECT customer_id, full_name, status
    FROM Customers
    WHERE city = p_city
    ORDER BY customer_id;
END //
DELIMITER ;
CALL sp_CustomersByCity('Da Nang');
```

Kết quả trên bộ dữ liệu mẫu:

customer_id	full_name	status
2	Tran Thi Binh	ACTIVE
6	Vu Thi Hoa	ACTIVE
15	Vo Van Tuan	ACTIVE
21	Truong Van Bao	ACTIVE
27	Vu Van Kien	ACTIVE

> **SQL Server vs MySQL:** tên tham số T-SQL luôn có `@` đứng trước; MySQL không có `@` và mọi người quen thêm tiền tố `p_` (hoặc `in_`) để **không trùng tên cột**. Nếu tham số tên `city` trùng cột `city`, MySQL sẽ hiểu là cột và câu `WHERE city = city` đúng với mọi dòng!

2.2. Tham số INPUT và OUTPUT

- **INPUT** (mặc định): giá trị truyền **vào** procedure.
- **OUTPUT**: procedure **trả giá trị ra** qua tham số để nơi gọi dùng tiếp (chỉ trả được 1 giá trị/tham số, khác với `SELECT` trả cả một bảng).

SQL Server (thêm từ khóa `OUTPUT` ở cả lúc khai báo lẫn lúc gọi):

```
GO
CREATE PROCEDURE sp_CountActiveAccounts @customer_id INT, @n INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    SELECT @n = COUNT(*) FROM Accounts
    WHERE customer_id = @customer_id AND status = 'ACTIVE';
END
GO
DECLARE @n INT;
EXEC sp_CountActiveAccounts 2, @n OUTPUT;
SELECT @n AS active_accounts;
```

MySQL (`IN` / `OUT` / `INOUT` viết ngay trước tên tham số; gọi bằng biến phiên `@n`):

```
DELIMITER //
CREATE PROCEDURE sp_CountActiveAccounts(IN p_customer_id INT, OUT p_n INT)
BEGIN
    SELECT COUNT(*) INTO p_n FROM Accounts
    WHERE customer_id = p_customer_id AND status = 'ACTIVE';
END //
DELIMITER ;
CALL sp_CountActiveAccounts(2, @n);
SELECT @n AS active_accounts;
```

Khách hàng số 2 có 3 tài khoản `ACTIVE` nên kết quả là `3`.

Tham số	Vai trò	T-SQL	MySQL
Đầu vào	Nơi gọi truyền giá trị vào	`@x INT`	`IN p_x INT`
Đầu ra	Procedure gán, nơi gọi đọc	`@x INT OUTPUT`	`OUT p_x INT`
Vào rồi ra	Truyền vào, procedure sửa, đọc lại	`@x INT OUTPUT`	`INOUT p_x INT`

> **Cảnh báo:** quên `OUTPUT` khi **gọi** trong T-SQL thì biến nhận về vẫn `NULL`. Ngoài ra tính "tổng" bằng `SUM()` mà khách hàng không có dòng nào cho kết quả `NULL`, phải bọc `COALESCE(SUM(...), 0)`. Bài 9.1b kiểm tra đúng bẫy này (khách hàng 4 chỉ có tài khoản đóng, khách hàng 10 không có tài khoản).

2.3. Biến cục bộ (DECLARE)

Biến giữ giá trị trung gian trong procedure.

	SQL Server	MySQL
Khai báo	`DECLARE @total DECIMAL(18,2);` (đặt ở bất cứ đâu, có thể gán ngay `= 0`)	`DECLARE v_total DECIMAL(18,2) DEFAULT 0;` (**phải** đặt ngay đầu khối `BEGIN`, trước các câu lệnh khác)
Gán giá trị	`SET @total = 10;` hoặc `SELECT @total = SUM(...) FROM ...;`	`SET v_total = 10;` hoặc `SELECT SUM(...) INTO v_total FROM ...;`

> **Lưu ý:** bẫy quen thuộc của MySQL là nhầm **biến cục bộ** (`DECLARE`, không có `@`, chỉ sống trong procedure) với **biến phiên** (`@ten`, sống suốt kết nối). Trong thân procedure hãy dùng biến cục bộ; biến phiên `@x` chỉ dùng ở nơi gọi để nhận tham số `OUT`.

2.4. Điều khiển luồng: IF, CASE, WHILE, "FOR"

IF ... ELSE — ví dụ phân mức theo số tài khoản đang hoạt động của khách hàng:

SQL Server

```
GO
CREATE PROCEDURE sp_AccountLevel @customer_id INT, @level NVARCHAR(20) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @n INT = (SELECT COUNT(*) FROM Accounts WHERE customer_id = @customer_id AND
        status = 'ACTIVE');
    IF @n >= 3
        SET @level = 'NHIEU TAI KHOAN';
    ELSE IF @n >= 1
        SET @level = 'CO TAI KHOAN';
    ELSE
        SET @level = 'CHUA CO TAI KHOAN';
END
GO
DECLARE @lv NVARCHAR(20);
EXEC sp_AccountLevel 2, @lv OUTPUT;
SELECT @lv AS muc;
```

MySQL (dùng `IF ... THEN ... ELSEIF ... ELSE ... END IF;`, chỉ dùng được **bên trong** procedure/trigger/function)

```
DELIMITER //
CREATE PROCEDURE sp_AccountLevel(IN p_customer_id INT, OUT p_level VARCHAR(20))
BEGIN
    DECLARE v_n INT DEFAULT 0;
    SELECT COUNT(*) INTO v_n FROM Accounts WHERE customer_id = p_customer_id AND status =
        'ACTIVE';
    IF v_n >= 3 THEN
        SET p_level = 'NHIEU TAI KHOAN';
    ELSEIF v_n >= 1 THEN
        SET p_level = 'CO TAI KHOAN';
    ELSE
        SET p_level = 'CHUA CO TAI KHOAN';
    END IF;
END //
DELIMITER ;
CALL sp_AccountLevel(2, @lv);
SELECT @lv AS muc;
```

Đổi số `2` thành `10` (khách hàng chưa có tài khoản) để thấy nhánh `ELSE`. Khối nhiều lệnh trong T-SQL dùng cặp `BEGIN ... END`.

CASE có hai dạng: biểu thức `CASE` trong `SELECT` (đã học ở Buổi 4, dùng được ở cả hai hệ) và **câu lệnh** `CASE` trong procedure (chỉ MySQL):

```sql
-- MySQL: câu lệnh CASE trong procedure
CASE
    WHEN v_total >= 1000000000 THEN SET p_tier = 'VIP';
    WHEN v_total >= 100000000  THEN SET p_tier = 'STANDARD';
    ELSE SET p_tier = 'BASIC';
END CASE;
```

Với T-SQL, trong procedure hãy dùng `IF...ELSE` hoặc gán bằng biểu thức: `SET @tier = CASE WHEN @total >= 1000000000 THEN 'VIP' WHEN @total >= 100000000 THEN 'STANDARD' ELSE 'BASIC' END;`

WHILE — lặp khi điều kiện còn đúng. Ví dụ cộng các số từ 1 đến n:

SQL Server

```sql
DECLARE @i INT = 1, @sum INT = 0, @n INT = 5;
WHILE @i <= @n
BEGIN
    SET @sum = @sum + @i;
    SET @i = @i + 1;
END
SELECT @sum AS tong;
```

MySQL (vòng lặp chỉ chạy được bên trong procedure)

```sql
DELIMITER //
CREATE PROCEDURE sp_SumTo(IN p_n INT, OUT p_sum INT)
BEGIN
    DECLARE i INT DEFAULT 1;
    SET p_sum = 0;
    WHILE i <= p_n DO
        SET p_sum = p_sum + i;
        SET i = i + 1;
    END WHILE;
END //
DELIMITER ;
CALL sp_SumTo(5, @s);
SELECT @s AS tong;
```

Cả hai cho `15` (tổng 1 + 2 + 3 + 4 + 5).

"FOR": cả T-SQL lẫn MySQL **không có** vòng `FOR i = 1 TO n`. Ta mô phỏng bằng `WHILE` (tăng biến đếm thủ công), hoặc trong MySQL dùng `LOOP ... LEAVE nhãn` / `REPEAT ... UNTIL`:

```mysql
-- MySQL: LOOP với nhãn, thoát bằng LEAVE (thay cho WHILE ở trên)
loop_label: LOOP
    IF i > p_n THEN LEAVE loop_label; END IF;
    SET p_sum = p_sum + i;
    SET i = i + 1;
END LOOP;
```

> **Cảnh báo:** SQL là ngôn ngữ **thao tác tập hợp**. Nếu bạn viết `WHILE` để duyệt từng khách hàng rồi `UPDATE` từng dòng, thường chậm hơn nhiều lần một câu `UPDATE ... WHERE` duy nhất. Chỉ dùng vòng lặp khi thật sự cần xử lý tuần tự (bài 9.1e là ví dụ mang tính luyện cú pháp).

2.5. RETURN và SELECT trong Procedure: ba cách "trả kết quả"

Cách	Trả gì	SQL Server	MySQL
Result set (`SELECT` cuối procedure)	Cả một bảng dữ liệu	Có	Có
Tham số OUTPUT/OUT	Vài giá trị đơn	Có	Có
RETURN giá trị	**Mã trạng thái nguyên** (thành công = 0)	`RETURN 0;` — nhận bằng `EXEC @rc = sp_x ...`	`RETURN` **không** mang giá trị trong procedure (chỉ dùng trong FUNCTION); thoát sớm bằng `LEAVE nhan`

Ví dụ T-SQL dùng `RETURN` để thoát sớm và trả mã: `0` nếu khách hàng tồn tại, `1` nếu không.

```tsql
GO
CREATE PROCEDURE sp_CheckCustomer @customer_id INT
AS
BEGIN
    SET NOCOUNT ON;
    IF NOT EXISTS (SELECT 1 FROM Customers WHERE customer_id = @customer_id)
        RETURN 1;          -- thoát sớm: không tìm thấy
    RETURN 0;              -- thành công
END
GO
DECLARE @rc INT;
EXEC @rc = sp_CheckCustomer 9999;
SELECT @rc AS return_code;
```

Kết quả `return_code = 1` vì khách hàng 9999 không tồn tại; đổi thành `EXEC @rc = sp_CheckCustomer 2;` sẽ được `0`.

MySQL thoát sớm khỏi procedure bằng cách gắn **nhãn** cho khối `BEGIN...END` rồi `LEAVE`:

```
DELIMITER //
CREATE PROCEDURE sp_CheckCustomer(IN p_customer_id INT, OUT p_rc INT)
main_block: BEGIN
    SET p_rc = 0;
    IF NOT EXISTS (SELECT 1 FROM Customers WHERE customer_id = p_customer_id) THEN
        SET p_rc = 1;
        LEAVE main_block;      -- thoát sớm, giống RETURN của T-SQL
    END IF;
END //
DELIMITER ;
CALL sp_CheckCustomer(9999, @rc);
SELECT @rc AS return_code;
```

> **Lưu ý:** mã `RETURN` của T-SQL chỉ là **số nguyên** để báo trạng thái (quy ước: 0 = thành công). Muốn trả dữ liệu thì dùng `OUTPUT` hoặc `SELECT`, đừng lạm dụng `RETURN` để trả số liệu nghiệp vụ.

> **Thực hành:** Vừa học xong tham số INPUT/OUTPUT, biến cục bộ, `IF` / `CASE` / `WHILE` và cách trả kết quả, hãy làm **bài 9.1** để tự viết 5 procedure: liệt kê khách hàng theo loại, tính tổng tài sản qua tham số OUTPUT, lấy TOP N, phân hạng bằng biến + `IF`, và khởi tạo `CustomerStats` bằng vòng lặp `WHILE`. Nhớ dòng `GO` (SQL Server) và `DELIMITER //` (MySQL).

- **Bài 9.1** — SP lấy danh sách khách hàng theo loại *(độ khó: dễ)*
- **Bài 9.1** — SP tính tổng tài sản khách hàng (tham số OUTPUT) *(độ khó: dễ)*
- **Bài 9.1** — SP lấy TOP N khách hàng giàu nhất *(độ khó: trung bình)*
- **Bài 9.1** — SP phân hạng tài sản (biến, IF/CASE) *(độ khó: trung bình)*
- **Bài 9.1** — SP khởi tạo CustomerStats bằng vòng lặp WHILE *(độ khó: trung bình)*

---

3. Gọi Stored Procedure

Viết xong procedure là mới một nửa; nơi gọi phải truyền đúng tham số và đọc đúng loại kết quả.

	SQL Server	MySQL
Không tham số	`EXEC sp_InitCustomerStats;`	`CALL sp_InitCustomerStats();`
Có tham số (theo thứ tự)	`EXEC sp_CustomersByCity 'Da Nang';`	`CALL sp_CustomersByCity('Da Nang');`
Theo tên tham số	`EXEC sp_CustomersByCity @city = 'Da Nang';`	(không hỗ trợ — truyền theo thứ tự)
Tham số OUTPUT	`EXEC sp 2, @n OUTPUT;`	`CALL sp(2, @n);`
Tham số có giá trị mặc định	`CREATE PROCEDURE sp_Top @n INT = 3 ...` rồi `EXEC sp_Top;`	(không hỗ trợ giá trị mặc định cho tham số)

Ví dụ giá trị mặc định (chỉ T-SQL): gọi không truyền tham số thì `@n` lấy giá trị `3`.

```
GO
CREATE PROCEDURE sp_RecentCustomers @n INT = 3
AS
BEGIN
    SET NOCOUNT ON;
    SELECT TOP (@n) customer_id, full_name, created_date
    FROM Customers
    ORDER BY created_date DESC, customer_id;
END
GO
EXEC sp_RecentCustomers;
```

3.1. Xử lý kết quả trả về

1. **Result set** đến trực tiếp ứng dụng/SSMS. Nếu cần xử lý tiếp **ngay trong SQL Server**, hứng vào bảng tạm bằng `INSERT ... EXEC`:

```
GO
CREATE PROCEDURE sp_RecentCustomers @n INT = 3
AS
BEGIN
    SET NOCOUNT ON;
    SELECT TOP (@n) customer_id, full_name, created_date
    FROM Customers
    ORDER BY created_date DESC, customer_id;
END
GO
CREATE TABLE #r (customer_id INT, full_name NVARCHAR(120), created_date DATE);
INSERT INTO #r EXEC sp_RecentCustomers @n = 2;      -- hứng kết quả của SP vào bảng tạm
SELECT COUNT(*) AS so_dong_hung_duoc FROM #r;
```

2. **OUTPUT / OUT** đọc từ biến (`@n`).
3. **Mã RETURN** (T-SQL) đọc bằng `EXEC @rc = ...`.

> **SQL Server vs MySQL:** MySQL không có `INSERT ... CALL` . Nếu cần dùng kết quả trong SQL, hãy để procedure ghi vào bảng tạm/bảng thật, hoặc dùng `FUNCTION` / `VIEW` thay cho procedure.

Xem lại mã nguồn procedure: SQL Server `EXEC sp_helptext 'sp_CustomersByCity';` (hoặc SSMS: *Programmability → Stored Procedures → Modify*). MySQL `SHOW CREATE PROCEDURE sp_CustomersByCity;` .

3.2. Procedure có ghi dữ liệu (UPDATE)

Procedure thường dùng để **ghi** dữ liệu theo một quy trình nghiệp vụ nhiều bước. Ví dụ đơn giản: đóng băng (chuyển `CLOSED`) các tài khoản `ACTIVE` của một khách hàng và trả về số tài khoản đã đổi. Số dòng bị ảnh hưởng: SQL Server `@@ROWCOUNT` (đọc **ngay** sau câu lệnh, câu lệnh kế tiếp sẽ ghi đè); MySQL `ROW_COUNT()` .

SQL Server

```
GO
CREATE PROCEDURE sp_FreezeCustomerAccounts @customer_id INT, @affected INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE Accounts SET status = 'CLOSED'
    WHERE customer_id = @customer_id AND status = 'ACTIVE';
    SET @affected = @@ROWCOUNT;          -- số dòng vừa bị UPDATE
END
GO
DECLARE @a INT;
EXEC sp_FreezeCustomerAccounts 2, @a OUTPUT;
SELECT @a AS so_tai_khoan_da_dong;
```

MySQL

```
DELIMITER //
CREATE PROCEDURE sp_FreezeCustomerAccounts(IN p_customer_id INT, OUT p_affected INT)
BEGIN
    UPDATE Accounts SET status = 'CLOSED'
    WHERE customer_id = p_customer_id AND status = 'ACTIVE';
    SET p_affected = ROW_COUNT();        -- số dòng vừa bị UPDATE
END //
DELIMITER ;
CALL sp_FreezeCustomerAccounts(2, @a);
SELECT @a AS so_tai_khoan_da_dong;
```

Khách hàng 2 có 3 tài khoản `ACTIVE` nên `so_tai_khoan_da_dong = 3`. Gọi lần thứ hai sẽ được `0` (không còn tài khoản nào `ACTIVE` để đổi): đây chính là cách procedure báo cho nơi gọi biết "đã làm được gì".

> **Cảnh báo:** vì thay đổi dữ liệu, hãy luôn `WHERE` chặt và thử trên dữ liệu test trước. Một `UPDATE` thiếu `WHERE` trong procedure sẽ sửa cả bảng mỗi lần ai đó gọi. Hãy kiểm tra tham số hợp lệ (trạng thái thuộc danh sách cho phép, khách hàng có tồn tại) **trước khi** ghi.

> **Thực hành:** Vừa học xong cách viết procedure có `UPDATE` và lấy số dòng bị ảnh hưởng, hãy làm **bài 9.2**: đổi trạng thái khách hàng có kiểm tra hợp lệ, tăng số dư cho khách VIP theo tỷ lệ, và khóa khách hàng "ngủ đông" kèm tham số OUTPUT báo số dòng đã đổi.

- **Bài 9.2** — SP cập nhật trạng thái khách hàng *(độ khó: dễ)*
- **Bài 9.2** — SP tăng số dư cho khách hàng VIP theo tỷ lệ *(độ khó: trung bình)*
- **Bài 9.2** — SP khóa khách hàng ngủ đông (OUTPUT số dòng ảnh hưởng) *(độ khó: khó)*

---

4. Trigger là gì?

Trigger (bộ kích hoạt) là một khối lệnh SQL **tự động chạy** khi có sự kiện `INSERT`, `UPDATE` hoặc `DELETE` xảy ra trên một bảng. Bạn **không gọi** nó bằng tay và nó gắn với **một bảng** cụ thể. Khác biệt cốt lõi với procedure: procedure "gọi thì chạy", trigger "có sự kiện thì chạy".

Hình dưới minh họa luồng của một trigger ghi nhật ký kiểm toán: người dùng chỉ thực hiện một câu `INSERT` / `UPDATE` / `DELETE`, còn việc ghi `AuditLog` được database tự động làm ngay sau đó.

Hình minh họa: Luồng chạy của Trigger (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: bắt đầu từ câu lệnh của người dùng ở bên trái, sự kiện xảy ra trên bảng gốc kích hoạt trigger, trigger chạy **cùng transaction** với câu lệnh gốc và ghi thêm dòng vào bảng nhật ký. Nếu câu lệnh gốc bị hoàn nguyên (`ROLLBACK`) thì phần trigger đã làm cũng bị hoàn nguyên theo.

4.1. Ứng dụng điển hình

- **Cập nhật tự động dữ liệu phái sinh:** mỗi khi thêm giao dịch khớp lệnh thì cộng dồn vào `CustomerStats`.
- **Audit log (nhật ký kiểm toán):** ai đổi trạng thái khách hàng, từ gì sang gì. Quy định của công ty chứng khoán về truy vết bắt buộc phải có.
- **Kiểm tra dữ liệu:** chặn số dư âm, chuẩn hóa email trước khi lưu.

4.2. Thời điểm kích hoạt: BEFORE / AFTER / INSTEAD OF

Loại	Chạy khi nào	MySQL	SQL Server
BEFORE	Trước khi ghi dòng vào bảng; có thể **sửa giá trị sắp ghi** hoặc từ chối	Có (`BEFORE INSERT/UPDATE/DELETE`)	**Không có**
AFTER	Sau khi dòng đã được ghi	Có	Có (`AFTER` hoặc `FOR`)
INSTEAD OF	**Thay thế** hoàn toàn thao tác gốc	Không có	Có (thường dùng trên VIEW hoặc để "xóa mềm")

> **SQL Server vs MySQL:** điểm khác biệt lớn nhất là **T-SQL không có BEFORE trigger**. Muốn "kiểm tra trước khi ghi" trong SQL Server bạn dùng `INSTEAD OF` (tự viết lệnh ghi) hoặc `AFTER` rồi `ROLLBACK` / `THROW` để hủy. Vì vậy các bài tập của buổi này dùng **AFTER** cho cả hai hệ để kết quả giống nhau.

4.3. Truy cập "dòng mới / dòng cũ": NEW/OLD và inserted/deleted

	MySQL	SQL Server
Đơn vị chạy	**Mỗi dòng** (`FOR EACH ROW`): trigger chạy N lần nếu câu lệnh chạm N dòng	**Mỗi câu lệnh**: trigger chạy **một lần** cho cả tập dòng bị ảnh hưởng
Dòng mới (INSERT/UPDATE)	`NEW.cột`	Bảng ảo `inserted` (có thể nhiều dòng)
Dòng cũ (UPDATE/DELETE)	`OLD.cột`	Bảng ảo `deleted` (có thể nhiều dòng)

Sự kiện	MySQL	SQL Server
INSERT	chỉ có `NEW`	chỉ có `inserted`
DELETE	chỉ có `OLD`	chỉ có `deleted`
UPDATE	`OLD` (cũ) và `NEW` (mới)	`deleted` (cũ) và `inserted` (mới), ghép bằng khóa chính

> **Cảnh báo:** bẫy lớn nhất khi chuyển từ MySQL sang T-SQL là trigger T-SQL chạy **một lần cho cả tập dòng**. Đừng viết như thể `inserted` luôn có đúng 1 dòng (`DECLARE @id = (SELECT customer_id FROM inserted)` chỉ lấy một dòng ngẫu nhiên, bỏ sót các dòng còn lại). Hãy luôn viết **theo kiểu tập hợp**: `UPDATE ... FROM ... JOIN inserted` , `INSERT ... SELECT ... FROM inserted` .

---

5. Viết Trigger

5.1. AFTER INSERT: tự tạo dòng thống kê cho khách hàng mới

Mỗi khách hàng cần đúng một dòng trong `CustomerStats` . Thay vì nhắc người nhập liệu, để trigger tự làm.

SQL Server

```
GO
CREATE TRIGGER trg_Customers_AfterInsert ON Customers
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO CustomerStats (customer_id, total_trade_value, trade_count)
    SELECT customer_id, 0, 0 FROM inserted;     -- xử lý được nhiều khách hàng cùng lúc
END
GO
INSERT INTO Customers (customer_id, customer_code, full_name, customer_type, city, email,
        status, created_date)
VALUES (101, 'KH900001', 'Nguyen Test Mot', 'INDIVIDUAL', 'Ha Noi', 'test1@example.com',
        'NEW', '2026-09-25'),
       (102, 'KH900002', 'Nguyen Test Hai', 'INDIVIDUAL', 'Ha Noi', 'test2@example.com',
        'NEW', '2026-09-25');
SELECT customer_id, total_trade_value, trade_count FROM CustomerStats WHERE customer_id >
        100 ORDER BY customer_id;
```

MySQL

```
DELIMITER //
CREATE TRIGGER trg_Customers_AfterInsert AFTER INSERT ON Customers
FOR EACH ROW
BEGIN
    INSERT INTO CustomerStats (customer_id, total_trade_value, trade_count)
    VALUES (NEW.customer_id, 0, 0);
END //
DELIMITER ;
INSERT INTO Customers (customer_id, customer_code, full_name, customer_type, city, email,
        status, created_date)
VALUES (101, 'KH900001', 'Nguyen Test Mot', 'INDIVIDUAL', 'Ha Noi', 'test1@example.com',
        'NEW', '2026-09-25'),
       (102, 'KH900002', 'Nguyen Test Hai', 'INDIVIDUAL', 'Ha Noi', 'test2@example.com',
        'NEW', '2026-09-25');
SELECT customer_id, total_trade_value, trade_count FROM CustomerStats WHERE customer_id >
        100 ORDER BY customer_id;
```

Kết quả: hai dòng thống kê `101` và `102` (giá trị 0.00 và 0) xuất hiện tự động dù ta chưa hề `INSERT` vào `CustomerStats`. Chú ý: một câu `INSERT` thêm **hai** khách hàng; trigger MySQL chạy hai lần (mỗi dòng một lần), trigger SQL Server chạy **một lần** và `inserted` chứa cả hai dòng.

5.2. AFTER INSERT / DELETE / UPDATE: giữ số liệu thống kê đồng bộ

Đây là nội dung các bài 9.3a, 9.3b, 9.3d. Dùng chung một ý tưởng: khách hàng sở hữu giao dịch được xác định qua chuỗi `Trades.order_id → Orders.account_id → Accounts.customer_id`.

Ví dụ ý tưởng khi thêm giao dịch (MySQL, mỗi dòng một lần):

```
UPDATE CustomerStats
SET total_trade_value = total_trade_value + NEW.quantity * NEW.price,
    trade_count       = trade_count + 1
WHERE customer_id = (SELECT a.customer_id
                     FROM Orders o JOIN Accounts a ON a.account_id = o.account_id
                     WHERE o.order_id = NEW.order_id);
```

Với T-SQL, trigger nhận cả tập `inserted`; nếu một câu `INSERT` thêm 2 giao dịch của cùng một khách hàng, phải **gom nhóm** (`GROUP BY customer_id`) trước khi cộng, nếu không chỉ một trong hai dòng được cộng (vì `UPDATE ... FROM ... JOIN` với nhiều dòng khớp chỉ áp dụng một dòng).

Kết quả mong đợi khi chèn 3 giao dịch (2 của khách hàng 22: 9.000.000 + 1.200.000; 1 của khách hàng 40: 27.000.000):

customer_id	total_trade_value	trade_count	Ghi chú
22	430725000.00	9	trước đó 420.525.000 và 7 giao dịch
40	27000000.00	1	trước đó 0 và 0 giao dịch

Với `DELETE` và `UPDATE` làm tương tự: khi xóa, **trừ** đi theo `deleted` / `OLD`; khi sửa khối lượng hoặc giá, cộng phần **chênh lệch** `(mới − cũ)`.

5.3. AFTER UPDATE: ghi AuditLog chỉ khi trạng thái thật sự đổi

Trigger `AFTER UPDATE` kích hoạt cho **mọi** câu `UPDATE` trên bảng, kể cả khi bạn chỉ sửa cột `city` hay gán lại đúng giá trị cũ. Vì vậy phải **so sánh** giá trị mới với cũ:

```
-- SQL Server: chỉ những dòng có status đổi
INSERT INTO AuditLog (customer_id, old_status, new_status)
SELECT i.customer_id, d.status, i.status
FROM inserted i JOIN deleted d ON d.customer_id = i.customer_id
WHERE d.status <> i.status;
```

```
-- MySQL
IF OLD.status <> NEW.status THEN
    INSERT INTO AuditLog (customer_id, old_status, new_status)
    VALUES (OLD.customer_id, OLD.status, NEW.status);
END IF;
```

> **Lưu ý:** nếu cột có thể `NULL`, `<>` cho kết quả `NULL` (không đúng, không sai) và bỏ sót thay đổi từ/đến `NULL`; khi đó dùng `NOT (OLD.x <=> NEW.x)` trong MySQL hoặc `ISNULL(d.x, '') <> ISNULL(i.x, '')` trong T-SQL. Cột `status` của Customers là `NOT NULL` nên dùng `<>` là đủ. Cột `changed_at` của AuditLog có giá trị mặc định (thời điểm hiện tại) nên không cần điền; vì "thời điểm hiện tại" thay đổi mỗi lần chạy, các bài chấm tự động **không in cột này**.

5.4. BEFORE trigger (MySQL) và INSTEAD OF trigger (SQL Server)

Tổng hợp bốn cơ chế thường gặp trong outline và ví dụ nghiệp vụ tương ứng:

Cơ chế	Dùng để	Ví dụ
`BEFORE INSERT` (MySQL)	Chuẩn hóa dữ liệu sắp ghi	Đổi email về chữ thường, mã khách hàng về chữ hoa
`AFTER INSERT`	Cập nhật dữ liệu phái sinh	Tạo dòng `CustomerStats`, cộng dồn giá trị giao dịch
`BEFORE UPDATE` (MySQL)	Từ chối dữ liệu sai	Chặn số dư tài khoản âm
`AFTER UPDATE`	Ghi vết thay đổi	Ghi `AuditLog` khi đổi trạng thái
`AFTER DELETE`	Trừ số liệu, lưu bản sao dòng bị xóa	Trừ giá trị giao dịch bị hủy khỏi `CustomerStats`

MySQL — BEFORE INSERT chuẩn hóa dữ liệu trước khi ghi (gán vào `NEW.cột`):

```
DELIMITER //
CREATE TRIGGER trg_Customers_BI BEFORE INSERT ON Customers FOR EACH ROW
BEGIN
    SET NEW.email = LOWER(TRIM(NEW.email));
    SET NEW.customer_code = UPPER(NEW.customer_code);
END //
DELIMITER ;
INSERT INTO Customers (customer_id, customer_code, full_name, customer_type, city, email,
        status, created_date)
VALUES (101, 'kh900001', 'Nguyen Test Mot', 'INDIVIDUAL', 'Ha Noi', '  Test.User@Example.COM
        ', 'NEW', '2026-09-25');
SELECT customer_id, customer_code, email FROM Customers WHERE customer_id = 101;
```

Email `' Test.User@Example.COM '` được lưu thành `test.user@example.com` và mã `kh900001` thành `KH900001`, dù câu `INSERT` ghi khác.

MySQL — BEFORE UPDATE từ chối dữ liệu sai bằng `SIGNAL` (gây lỗi, câu lệnh bị hủy):

```
CREATE TRIGGER trg_Accounts_NoNegative BEFORE UPDATE ON Accounts FOR EACH ROW
BEGIN
    IF NEW.balance < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'So du khong duoc am';
    END IF;
END
```

Câu `UPDATE Accounts SET balance = -1 ...` sẽ báo lỗi `So du khong duoc am` và không dòng nào bị đổi.

SQL Server — INSTEAD OF DELETE: "xóa mềm" khách hàng (không cho xóa thật, chỉ đánh dấu `INACTIVE`):

```
GO
CREATE TRIGGER trg_Customers_SoftDelete ON Customers
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE c SET status = 'INACTIVE'
    FROM Customers c JOIN deleted d ON d.customer_id = c.customer_id;
END
GO
DELETE FROM Customers WHERE customer_id = 10;
SELECT customer_id, status FROM Customers WHERE customer_id = 10;
```

Dòng `10 INACTIVE` vẫn còn trong bảng: lệnh `DELETE` đã bị trigger thay thế bằng `UPDATE`.

SQL Server — AFTER + ROLLBACK/THROW thay cho BEFORE kiểm tra dữ liệu:

```
GO
CREATE TRIGGER trg_Accounts_NoNegative ON Accounts
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM inserted WHERE balance < 0)
    BEGIN
        ROLLBACK TRANSACTION;                    -- hủy cả câu UPDATE vừa chạy
        THROW 50001, 'So du khong duoc am', 1;
    END
END
GO
```

5.5. Quản lý trigger

Việc	SQL Server	MySQL
Liệt kê	`SELECT name FROM sys.triggers;`	`SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_TIMING FROM INFORMATION_SCHEMA.TRIGGERS WHERE TRIGGER_SCHEMA = DATABASE();`
Xóa	`DROP TRIGGER trg_Customers_AfterInsert;`	`DROP TRIGGER trg_Customers_AfterInsert;`
Tạm tắt	`DISABLE TRIGGER trg_x ON Customers;`	(không có — phải xóa và tạo lại)

5.6. Sai lầm thường gặp với trigger

1. **Đệ quy / chuỗi trigger:** trigger A ghi bảng B, trigger của B lại ghi bảng A... khó lường. MySQL cấm trigger sửa **chính bảng** đang kích hoạt nó.
2. **Trigger "ngầm":** người viết câu `INSERT` không biết còn có trigger chạy nên khó gỡ lỗi. Luôn tài liệu hóa.
3. **Coi trigger chạy 1 dòng (T-SQL):** như đã cảnh báo ở mục 4.3.
4. **Việc nặng trong trigger:** trigger chạy trong cùng transaction với thao tác gốc; xử lý chậm sẽ làm chậm mọi lệnh đặt/khớp lệnh và giữ khóa lâu.
5. **Import hàng loạt:** nạp 1 triệu dòng kích hoạt trigger 1 triệu lần (MySQL). Cân nhắc tắt trigger tạm thời (`DISABLE TRIGGER`) khi migrate, rồi tính lại số liệu.

> **Thực hành:** Vừa học xong cả trigger `AFTER INSERT` / `DELETE` / `UPDATE` với `NEW` / `OLD` và `inserted` / `deleted` , hãy làm **bài 9.3**: giữ `CustomerStats` đồng bộ khi thêm, xóa, sửa giao dịch và ghi `AuditLog` khi khách hàng đổi trạng thái. Hệ thống chấm sẽ tự kích hoạt trigger của bạn bằng các lệnh DML rồi so sánh dữ liệu, kể cả trường hợp nhiều dòng trong một câu lệnh.

- **Bài 9.3** — Trigger cập nhật CustomerStats khi thêm giao dịch *(độ khó: trung bình)*
- **Bài 9.3** — Trigger cập nhật lại CustomerStats khi xóa giao dịch *(độ khó: trung bình)*
- **Bài 9.3** — Trigger ghi AuditLog khi đổi trạng thái khách hàng *(độ khó: trung bình)*
- **Bài 9.3** — Trigger AFTER UPDATE điều chỉnh CustomerStats (NEW/OLD) *(độ khó: khó)*

---

6. Best Practices

6.1. Khi nào dùng Stored Procedure, khi nào dùng Application Code?

Nên đặt trong Stored Procedure / Trigger	Nên đặt trong code ứng dụng
Thao tác **thuần dữ liệu**, nhiều bước, cần chạy sát CSDL (chuyển tiền, đóng phiên, đối soát)	Logic nghiệp vụ phức tạp, nhiều rẽ nhánh, cần thư viện ngoài (gọi API, gửi mail, mã hóa)
Quy tắc **bắt buộc mọi hệ thống phải tuân thủ** dù ai truy cập (audit log)	Logic hay thay đổi, cần triển khai/rollback nhanh cùng với bản phát hành phần mềm
Cần **bảo mật theo quyền EXECUTE** (không cho truy cập bảng trực tiếp)	Cần **unit test** tự động, CI/CD, debug dễ bằng IDE
Xử lý **khối lượng lớn** mà chuyển dữ liệu ra ứng dụng sẽ tốn mạng	Xử lý phụ thuộc trạng thái phiên người dùng, giao diện

> **Mẹo:** nguyên tắc thực tế là **"thao tác dữ liệu ở database, quyết định nghiệp vụ phức tạp ở ứng dụng"**. Đừng nhồi toàn bộ nghiệp vụ vào procedure (khó test, khó dùng chung hệ quản trị khác, khó mở rộng ngang); nhưng cũng đừng để ứng dụng gửi hàng nghìn câu SQL nhỏ mỗi thao tác.

6.2. Quản lý version, testing, documentation

- **Version:** lưu script `CREATE PROCEDURE/TRIGGER` trong Git (một file/đối tượng), đánh số migration; không sửa trực tiếp trên production bằng tay. Dùng `CREATE OR ALTER` (T-SQL) hoặc `DROP ... IF EXISTS` (MySQL) để script chạy lại nhiều lần được.
- **Testing:** chạy procedure trên **dữ liệu thử**, kiểm cả **trường hợp biên** như buổi này: khách hàng không có tài khoản (`NULL`), đồng hạng, tham số không hợp lệ, số dòng bằng 0. Viết script kiểm thử: chuẩn bị dữ liệu, gọi, `SELECT` trạng thái, so với kỳ vọng. Đó chính là cách các bài chấm "verify" của Codelab hoạt động.
- **Documentation:** ghi chú đầu procedure (mục đích, tham số, giá trị trả về, ví dụ gọi, tác giả/ngày sửa). Đặt tên nhất quán: `sp_<Động từ><Đối tượng>` , `trg_<Bảng>_<Thời điểm><Sự kiện>` .
- **Xử lý lỗi:** T-SQL dùng `TRY ... CATCH` , MySQL dùng `DECLARE ... HANDLER` ; procedure ghi dữ liệu nên bọc trong transaction (sẽ học ở Buổi 10).
- **Hiệu năng:** đặt `SET NOCOUNT ON` (T-SQL); không viết `SELECT *` ; tránh con trỏ/vòng lặp khi câu lệnh tập hợp làm được.

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
SQL Server: quên dòng `GO` trước `CREATE PROCEDURE/TRIGGER`	Lỗi "CREATE PROCEDURE must be the only statement in the batch"	Bắt đầu script bằng `GO` , ngăn các khối bằng `GO`
MySQL: quên `DELIMITER //` (hoặc quên trả lại `DELIMITER ;`)	Lỗi cú pháp ngay dòng `;` đầu tiên trong thân	Bao khối bằng `DELIMITER //` ... `END //` `DELIMITER ;`
Quên `OUTPUT` khi gọi (T-SQL)	Biến nhận về luôn `NULL`	`EXEC sp 2, @n OUTPUT;`
Đặt tên tham số MySQL trùng tên cột	`WHERE city = city` đúng với mọi dòng	Thêm tiền tố `p_` cho tham số, `v_` cho biến
Tính `SUM()` mà không có dòng nào	Trả `NULL` thay vì 0	`COALESCE(SUM(...), 0)`
Đọc `@@ROWCOUNT` sau khi đã chạy thêm lệnh khác	Số dòng sai (bị ghi đè)	Gán ngay sau câu `UPDATE` : `SET @affected = @@ROWCOUNT;`
`UPDATE` trong procedure thiếu `WHERE` hoặc không kiểm tra tham số	Sửa cả bảng mỗi lần gọi	`WHERE` chặt, kiểm tra tham số hợp lệ trước khi ghi
Dùng `WHILE` duyệt từng dòng thay cho một câu tập hợp	Chậm hơn hàng chục đến hàng trăm lần	Viết một câu `UPDATE/INSERT ... SELECT`
Trigger T-SQL giả định `inserted` chỉ có 1 dòng	Bỏ sót các dòng còn lại khi `INSERT` nhiều dòng	Viết theo tập hợp: `JOIN inserted` , `GROUP BY` trước khi cộng
Trigger `AFTER UPDATE` không so sánh giá trị cũ/mới	Ghi log cả khi không có thay đổi	So sánh `OLD.x <> NEW.x` / `d.x <> i.x`
Trigger ghi ngược lên chính bảng hoặc gọi vòng tròn	Lỗi (MySQL) hoặc đệ quy khó kiểm soát	Thiết kế một chiều, tài liệu hóa mọi trigger

Tóm tắt

Chủ đề	Ghi nhớ
Stored Procedure	Khối lệnh đặt tên, lưu trong CSDL; lợi ích: tái sử dụng, bảo mật, hiệu năng, kiểm soát
Tạo / gọi	T-SQL: `GO` + `CREATE PROCEDURE ... AS BEGIN ... END` / `EXEC` ; MySQL: `DELIMITER //` ... / `CALL`
Tham số	INPUT mặc định; OUTPUT (T-SQL: `@x INT OUTPUT` ; MySQL: `OUT p_x INT`)
Biến	`DECLARE` ; T-SQL gán bằng `SET` / `SELECT @x = ...` , MySQL `SET` / `SELECT ... INTO`
Điều khiển	`IF` , `CASE` , `WHILE` ; không có `FOR` (dùng `WHILE` / `LOOP` / `REPEAT`)
Trả kết quả	Result set (`SELECT`), tham số OUTPUT, mã `RETURN` (chỉ T-SQL)
Số dòng bị ảnh hưởng	T-SQL `@@ROWCOUNT` ; MySQL `ROW_COUNT()`
Trigger	Tự chạy khi INSERT/UPDATE/DELETE; MySQL `FOR EACH ROW` + `NEW` / `OLD` ; SQL Server theo câu lệnh + `inserted` / `deleted`
Thời điểm	MySQL: BEFORE/AFTER; SQL Server: AFTER và INSTEAD OF (không có BEFORE)
Best practice	Thao tác dữ liệu ở CSDL, nghiệp vụ phức tạp ở ứng dụng; quản lý version, test cả trường hợp biên, tài liệu hóa

Gợi ý mở rộng

- Viết procedure `sp_RecalcCustomerStats` tính lại **toàn bộ** `CustomerStats` từ bảng `Trades` bằng một câu lệnh tập hợp, dùng để đối chiếu với số liệu do trigger duy trì (phát hiện lệch).
- Thêm cột `updated_by` vào AuditLog và ghi người thay đổi (T-SQL `SUSER_SNAME()` , MySQL `CURRENT_USER()`).
- Tìm hiểu **User-Defined Function** (hàm trả về giá trị/bảng dùng ngay trong `SELECT`) và khác biệt với Stored Procedure.
- Tìm hiểu xử lý lỗi `TRY ... CATCH` (T-SQL) / `DECLARE ... HANDLER` (MySQL), chuẩn bị cho phần Transaction ở Buổi 10.

Bài tập thực hành của buổi

- **Bài 9.1** — SP lấy danh sách khách hàng theo loại *(độ khó: dễ)*
- **Bài 9.1** — SP tính tổng tài sản khách hàng (tham số OUTPUT) *(độ khó: dễ)*
- **Bài 9.1** — SP lấy TOP N khách hàng giàu nhất *(độ khó: trung bình)*
- **Bài 9.1** — SP phân hạng tài sản (biến, IF/CASE) *(độ khó: trung bình)*
- **Bài 9.1** — SP khởi tạo CustomerStats bằng vòng lặp WHILE *(độ khó: trung bình)*
- **Bài 9.2** — SP cập nhật trạng thái khách hàng *(độ khó: dễ)*
- **Bài 9.2** — SP tăng số dư cho khách hàng VIP theo tỷ lệ *(độ khó: trung bình)*
- **Bài 9.2** — SP khóa khách hàng ngủ đông (OUTPUT số dòng ảnh hưởng) *(độ khó: khó)*
- **Bài 9.3** — Trigger cập nhật CustomerStats khi thêm giao dịch *(độ khó: trung bình)*

- **Bài 9.3** — Trigger cập nhật lại CustomerStats khi xóa giao dịch *(độ khó: trung bình)*
- **Bài 9.3** — Trigger ghi AuditLog khi đổi trạng thái khách hàng *(độ khó: trung bình)*
- **Bài 9.3** — Trigger AFTER UPDATE điều chỉnh CustomerStats (NEW/OLD) *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: tạo, chạy và quản lý Procedure, Trigger

1. Tạo Stored Procedure trong SSMS. Gõ vào cửa sổ truy vấn mới, nhớ dòng `GO` (như đã nói ở đầu buổi):

```
USE hnx_securities;
GO
CREATE OR ALTER PROCEDURE sp_KhachHangTheoThanhPho
  @thanh_pho NVARCHAR(30)
AS
BEGIN
  SET NOCOUNT ON;
  SELECT customer_id, full_name, city
  FROM Customers
  WHERE city = @thanh_pho
  ORDER BY customer_id;
END
GO
```

Bấm **F5**. Trong Object Explorer mở **hnx_securities** → **Programmability** → **Stored Procedures** (bấm **Refresh**): `dbo.sp_KhachHangTheoThanhPho` đã xuất hiện.

2. Chạy procedure theo hai cách.

- Bằng lệnh: `EXEC sp_KhachHangTheoThanhPho @thanh_pho = N'Ha Noi';`
- Bằng giao diện: chuột phải procedure → **Execute Stored Procedure...**, nhập giá trị vào cột *Value*, bấm **OK**. SSMS tự sinh lệnh `EXEC` tương ứng, hãy đọc để học cú pháp.

3. Xem và sửa mã nguồn. Chuột phải procedure → **Modify** (mở bản `ALTER PROCEDURE`), hoặc **Script Stored Procedure as** → **CREATE To** → **New Query Editor Window**. Xem nhanh bằng lệnh: `EXEC sp_helptext 'sp_KhachHangTheoThanhPho';`

4. Tạo và quan sát Trigger. Trigger của bảng nằm trong **Tables** → **bảng** → **Triggers**. Ví dụ ghi log khi đổi trạng thái khách hàng:

```
CREATE OR ALTER TRIGGER trg_Customers_GhiLog ON Customers
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO AuditLog (customer_id, old_status, new_status)
  SELECT d.customer_id, d.status, i.status
  FROM deleted d JOIN inserted i ON i.customer_id = d.customer_id
  WHERE ISNULL(d.status, '') <> ISNULL(i.status, '');
END
GO
```

Thử kích hoạt trong một giao dịch rồi hoàn nguyên để không đổi dữ liệu mẫu:

```
BEGIN TRANSACTION;
UPDATE Customers SET status = 'INACTIVE' WHERE customer_id = 1;
SELECT TOP 5 * FROM AuditLog ORDER BY log_id DESC;     -- thấy dòng log vừa sinh
ROLLBACK;
```

5. Tắt/bật trigger khi gỡ lỗi. `DISABLE TRIGGER trg_Customers_GhiLog ON Customers;` rồi `ENABLE TRIGGER ...`, hoặc chuột phải trigger → **Disable/Enable**. Xóa bằng `DROP TRIGGER trg_Customers_GhiLog;`. Nhớ **dọn trigger và procedure thử** để không ảnh hưởng bài khác.

Với MySQL Workbench: procedure nằm ở **Schemas** → **hnx_securities** → **Stored Procedures**, chuột phải → *Create Stored Procedure...*; trong script nhớ `DELIMITER //` ... `//` `DELIMITER ;`; gọi bằng `CALL`.

Tự kiểm tra: viết lại **bài 9.1** (procedure tính tổng tài sản có tham số OUTPUT) trong SSMS, chạy bằng cả `EXEC` và **Execute Stored Procedure...**, rồi nộp trên Codelab.

Buổi 10 — Index & Transaction + Review toàn bộ khóa

Về mục lục

Đây là buổi cuối của khóa. Nửa đầu học hai kỹ thuật quyết định **tốc độ** và **độ tin cậy** của một hệ thống chứng khoán: **Index** (tìm nhanh trong hàng triệu lệnh, giao dịch) và **Transaction** (chuyển tiền không bao giờ "bốc hơi" giữa chừng). Nửa sau là **ôn tập toàn bộ khóa** và làm **Capstone project**: bộ báo cáo phân tích khách hàng cho phòng chăm sóc khách hàng.

> **Lưu ý:** bộ dữ liệu là **hnx_securities** (xem mục "Bộ dữ liệu mẫu của khóa học" ở Buổi 1). "Hôm nay" cố định là **2026-09-25**. Buổi này có nhiều phần **không chấm tự động** (execution plan, hai phiên đồng thời, kết nối Excel); các phần đó được trình bày thành các bước làm tuần tự để bạn tự thực hành trên SSMS/Workbench và Excel của công ty.

Mục tiêu buổi học

Sau buổi này bạn có thể:

- Giải thích index là gì, B-Tree hoạt động thế nào, lợi ích (tăng tốc `SELECT`) và chi phí (chậm ghi, tốn dung lượng); biết khi nào nên và không nên tạo index.
- Phân biệt các loại index: PRIMARY KEY (clustered), UNIQUE, composite (quy tắc cột trái nhất), covering (`INCLUDE`), FULLTEXT; nắm khác biệt SQL Server / MySQL (NULL trong unique index, index khóa ngoại).
- Tạo, xem, xóa index (`CREATE INDEX`, `DROP INDEX`, `sys.indexes` / `INFORMATION_SCHEMA.STATISTICS`); đọc execution plan trên SSMS và `EXPLAIN` trên MySQL; hiểu index hints / `FORCE INDEX`.
- Viết transaction (`BEGIN` / `START TRANSACTION`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`) tôn trọng ACID và xây procedure chuyển tiền hoàn nguyên khi lỗi.
- Hiểu isolation level (dirty / non-repeatable / phantom read), lock và deadlock cùng cách phòng tránh.
- Viết truy vấn hiệu quả (sargable), biết công cụ monitoring, biết kết nối và trích xuất dữ liệu SQL vào Excel, và hoàn thành Capstone.

Bản đồ buổi học

Buổi học 180 phút: khoảng 33% lý thuyết + demo, 67% thực hành. Bài tập nằm xen ngay trong bài giảng; Capstone đặt ở cuối buổi.

Phần	Nội dung	Hình thức	Thời lượng gợi ý
1	Index là gì? B-Tree, lợi ích, nhược điểm, thử nghiệm có/không index	Lý thuyết + demo	7 phút
2	Các loại Index, composite, FULLTEXT, khi nào tạo index	Lý thuyết + demo	7 phút
3	Tạo và quản lý Index, `EXPLAIN`, index hints, tối ưu truy vấn chậm	Lý thuyết + demo	10 phút
	Thực hành bài 10.1 (4 bài) và **bài 10.2** (1 bài), gồm chữa bài	Thực hành	24 + 10 phút
4	Transaction: ACID, `COMMIT` / `ROLLBACK` / `SAVEPOINT`, isolation, deadlock	Lý thuyết + mô phỏng	14 phút
	Thực hành bài 10.3 (3 bài, gồm chữa bài)	Thực hành	26 phút
5	Hiệu năng: truy vấn hiệu quả, lỗi thường gặp, monitoring	Lý thuyết	5 phút
6	Kết nối và trích xuất dữ liệu SQL từ Excel	Hướng dẫn từng bước	9 phút
7	Review toàn khóa và **Capstone** (bài 10.4, 8 bài; làm cá nhân hoặc theo cặp, đổi chéo chữa bài; bài khó 10.4f, 10.4h có thể hoàn thiện sau giờ học)	Ôn tập + dự án	8 + 60 phút
	Tổng	Lý thuyết + demo ~60 phút, thực hành ~120 phút (67%)	**180 phút**

---

1. Index (chỉ mục) là gì?

1.1. Khái niệm

Index là một cấu trúc dữ liệu phụ được database lưu **bên cạnh** bảng để **tăng tốc tìm kiếm**. Ví dụ đời thường: mục lục cuối sách. Muốn tìm "Trái phiếu", bạn tra mục lục (đã sắp thứ tự chữ cái) rồi nhảy thẳng tới trang, thay vì lật từng trang từ đầu.

Không có index, để tìm `WHERE account_id = 42` database phải **quét toàn bộ bảng (table scan)** — đọc mọi dòng. Có index trên `account_id`, database đi thẳng tới đúng dòng cần (**index seek**).

> **Ví dụ nghiệp vụ:** bảng `Trades` của một công ty chứng khoán có thể lên tới hàng chục triệu dòng sau vài năm. Màn hình "tra cứu giao dịch của khách hàng X" mà quét cả bảng sẽ mất hàng chục giây; có index đúng chỉ mất vài mili-giây.

1.2. Index hoạt động thế nào? (B-Tree)

Loại index phổ biến nhất là **B-Tree (cây cân bằng)** — dùng cho cả SQL Server và MySQL InnoDB. Hình dưới so sánh việc tìm một giá trị khi **không có index** (đọc lần lượt từng dòng) với khi **có index B-Tree** (đi từ gốc xuống lá qua vài bước).

Hình minh họa: Cấu trúc cây B-Tree của index (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: nút gốc (root) chứa các mốc để rẽ nhánh, nút trung gian thu hẹp tiếp, nút lá (leaf) chứa giá trị khóa **đã sắp xếp** cùng con trỏ tới dòng dữ liệu. Mỗi bước đi xuống loại bỏ phần lớn dữ liệu còn lại.

- Các khóa được **sắp xếp sẵn**: bảng 1 triệu dòng thường chỉ cần **3–4 lần đọc trang** thay vì đọc cả bảng.
- Nút lá được nối với nhau theo thứ tự nên tìm **khoảng** (`BETWEEN` , `>=` , `<`) và `ORDER BY` theo cột index cũng nhanh.
- Khi `INSERT/UPDATE/DELETE` , database phải **cập nhật cả cây** để giữ thứ tự (đôi khi tách/gộp nút).

1.3. Lợi ích và nhược điểm

	Nội dung
Lợi ích	`SELECT ... WHERE` , `JOIN` , `ORDER BY` , `GROUP BY` theo cột có index nhanh hơn rất nhiều; index `UNIQUE` còn **bảo đảm không trùng**
Nhược điểm 1	**Chậm** `INSERT/UPDATE/DELETE` : mỗi thay đổi phải cập nhật bảng **và mọi index** của bảng đó
Nhược điểm 2	**Tốn dung lượng** (đĩa và bộ nhớ đệm), đôi khi bằng cả bảng
Nhược điểm 3	Index thừa làm optimizer khó chọn và tăng chi phí bảo trì

> **Mẹo:** quy tắc nhớ nhanh: **index giúp đọc, làm chậm ghi**. Bảng tra cứu nhiều (Customers, Securities) thường đáng có nhiều index; bảng ghi cực nhiều (Trades, Orders trong phiên khớp lệnh) phải cân nhắc kỹ từng index.

1.4. Thử nghiệm: có Index và không có Index

Bảng mẫu chỉ có vài chục dòng nên chênh lệch không thấy rõ. Ta tự tạo một bảng lớn hơn (khoảng **310.000 dòng**) từ tích chéo các bảng có sẵn (đây là phần **thực hành thủ công trên SSMS/Workbench**, không chạy thử trên trang).

SQL Server

```sql
SELECT t.symbol, t.quantity, t.price,
       DATEADD(DAY, o.order_id % 400, '2025-01-01') AS trade_date, o.order_id AS ref_id
INTO BigTrades
FROM Trades t CROSS JOIN Orders o CROSS JOIN Accounts a
WHERE a.account_id <= 30;                    -- 86 x 120 x 30 = 309.600 dòng

SET STATISTICS IO ON;                        -- xem số trang phải đọc trong tab Messages
SELECT COUNT(*), SUM(quantity) FROM BigTrades WHERE trade_date = '2025-03-01';
-- Table 'BigTrades'. Scan count 1, logical reads 1745   <- quét cả bảng

CREATE INDEX ix_Big_date ON BigTrades (trade_date) INCLUDE (quantity);
SELECT COUNT(*), SUM(quantity) FROM BigTrades WHERE trade_date = '2025-03-01';
-- Table 'BigTrades'. Scan count 1, logical reads 14     <- chỉ 14 trang
```

MySQL

```sql
CREATE TABLE BigTrades (symbol VARCHAR(10), quantity INT, price DECIMAL(12,2), trade_date
        DATE, ref_id INT);
INSERT INTO BigTrades
SELECT t.symbol, t.quantity, t.price, DATE_ADD('2025-01-01', INTERVAL (o.order_id % 400)
        DAY), o.order_id
FROM Trades t CROSS JOIN Orders o CROSS JOIN Accounts a WHERE a.account_id <= 30;

EXPLAIN SELECT COUNT(*), SUM(quantity) FROM BigTrades WHERE trade_date = '2025-03-01';
-- type = ALL, key = NULL,        rows ~ 309.000    <- quét toàn bộ

CREATE INDEX ix_Big_date ON BigTrades (trade_date);
EXPLAIN SELECT COUNT(*), SUM(quantity) FROM BigTrades WHERE trade_date = '2025-03-01';
-- type = ref, key = ix_Big_date, rows = 2580         <- tra thẳng ~2.580 dòng
```

Tóm tắt kết quả đo:

	Không index	**Có index**
SQL Server (logical reads)	1.745 trang	14 trang
MySQL (`type` / `rows`)	`ALL` / ~309.000	`ref` / 2.580
Kết quả truy vấn	2580 dòng, tổng khối lượng 2.430.000	**giống hệt**

Kết quả truy vấn vẫn như cũ: **index không đổi kết quả, chỉ đổi cách đến kết quả.**

> **Cảnh báo:** hai điều rút ra khi thử nghiệm (đã kiểm chứng khi soạn bài). (1) Trên SQL Server, nếu chỉ tạo `INDEX (trade_date)` **không** có `INCLUDE (quantity)` , optimizer vẫn **chọn quét bảng** vì mỗi dòng tìm được phải "tra ngược" (key lookup) về bảng để lấy `quantity` , và 2.580 lần tra ngược tốn hơn cả bảng (1.745 trang). Index chỉ được dùng khi **có lợi**; index có đủ mọi cột truy vấn cần gọi là **covering index** (chỉ mục phủ). (2) Với bảng nhỏ (vài trăm dòng) optimizer luôn quét bảng vì rẻ hơn; đừng kết luận "index vô dụng" khi đo trên dữ liệu bé.

Xong nhớ dọn: `DROP TABLE BigTrades;`. Để **xem chi phí ghi**: chèn 10.000 dòng vào `BigTrades` khi có 0 index, 1 index, 3 index rồi so thời gian `SET STATISTICS TIME ON`; mỗi index thêm vào làm thời gian tăng rõ rệt.

---

2. Các loại Index

Loại	Đặc điểm	SQL Server	MySQL (InnoDB)
PRIMARY KEY index	Tự sinh khi khai báo khóa chính; không NULL, không trùng	Mặc định là **clustered** (sắp xếp vật lý cả bảng theo khóa)	Luôn là **clustered index**; mọi index phụ đều chứa thêm khóa chính
UNIQUE index	Không cho trùng giá trị (kiêm ràng buộc `UNIQUE`)	**Chỉ cho một NULL**; muốn nhiều NULL dùng filtered index (`WHERE col IS NOT NULL`)	Cho phép **nhiều NULL**
Non-clustered / index thường	Không ràng buộc; chỉ để tìm nhanh	`CREATE INDEX`	`CREATE INDEX`
COMPOSITE index	Nhiều cột, **thứ tự cột quan trọng**	`(a, b)`	`(a, b)`
Covering (INCLUDE)	Kèm cột phụ ở nút lá để khỏi tra ngược	`INCLUDE (col)`	Không có `INCLUDE`, đưa cột vào cuối composite
Filtered index	Chỉ index một phần dòng	`WHERE ...`	Không có (dùng generated column)
FULLTEXT index	Tìm kiếm **từ/cụm từ** trong văn bản dài	Cần *Full-Text Catalog* rồi `CREATE FULLTEXT INDEX`, dùng `CONTAINS` / `FREETEXT`	`CREATE FULLTEXT INDEX`, dùng `MATCH(col) AGAINST('...')`

2.1. Composite index và quy tắc "cột trái nhất" (leftmost prefix)

Composite `(account_id, order_date)` giống danh bạ sắp xếp theo *họ* rồi *tên*:

Điều kiện `WHERE`	Dùng được index `(account_id, order_date)`?	Vì sao
`account_id = 42`	Có	Cột đầu có mặt
`account_id = 42 AND order_date >= '2026-09-01'`	**Có, tốt nhất**	Đẳng thức rồi khoảng, đúng thứ tự
`order_date >= '2026-09-01'`	Không (hoặc rất kém)	Thiếu cột đầu, giống tìm theo tên mà không biết họ

Quy tắc thứ tự cột: cột dùng điều kiện `=` đứng **trước**, cột dùng khoảng (`>=`, `BETWEEN`) đứng **sau**; cột chọn lọc cao (ít trùng) ưu tiên trước nếu cùng loại điều kiện.

2.2. FULLTEXT index

`LIKE '%Ngan Hang%'` (có `%` ở đầu) không dùng được index B-Tree. Với văn bản dài, dùng full-text.

MySQL

```
CREATE FULLTEXT INDEX ft_customers_name ON Customers (full_name);
SELECT customer_id, full_name FROM Customers
WHERE MATCH(full_name) AGAINST('Ngan Hang' IN NATURAL LANGUAGE MODE)
ORDER BY customer_id;
```

Kết quả gồm bốn ngân hàng: `31 Ngan Hang TMCP Dau Tu Viet`, `32 Ngan Hang Nong Nghiep Phat Trien`, `37 Ngan Hang Quan Doi Mien Bac`, `40 Ngan Hang Ban Le Sai Gon`.

SQL Server cần Full-Text Search (một thành phần cài thêm):

```
CREATE FULLTEXT CATALOG ft_cat AS DEFAULT;
CREATE FULLTEXT INDEX ON Customers (full_name) KEY INDEX <tên PK>;   -- tên PK xem bằng
        sys.indexes
SELECT customer_id, full_name FROM Customers WHERE CONTAINS(full_name, '"Ngan Hang"');
```

> **SQL Server vs MySQL:** Codelab không chấm phần full-text trên SQL Server; bạn thực hành trên SSMS của công ty nếu đã bật tính năng.

2.3. Khi nào nên tạo Index?

Nên tạo cho	Cẩn thận / không nên với
Cột **khóa ngoại** dùng để `JOIN` (`Accounts.customer_id`, `Orders.account_id`, `Trades.order_id`)	Bảng rất nhỏ (optimizer luôn quét bảng)
Cột thường xuất hiện trong `WHERE`, `ORDER BY`, `GROUP BY` của truy vấn chạy nhiều lần (`trade_date`, `order_date`)	Cột có **ít giá trị khác nhau** (`status` 3 giá trị, `side` BUY/SELL): index ít chọn lọc, optimizer bỏ qua
Cột cần **bảo đảm duy nhất** (email, mã khách hàng)	Bảng **ghi nhiều, đọc ít**
Cột dùng để sắp xếp/phân trang thường xuyên	Cột bị bọc hàm trong `WHERE` (`YEAR(trade_date)`): index vô dụng nếu không viết lại truy vấn

> **SQL Server vs MySQL:** SQL Server **không tự tạo** index cho khóa ngoại (chỉ tạo cho PK/UNIQUE) nên bạn phải tự tạo; MySQL InnoDB **tự tạo** index cho khóa ngoại (`SHOW INDEX FROM Orders` cho thấy index tên `account_id`, `symbol`). Đừng tạo quá nhiều index: mỗi index là một cây phải cập nhật.

---

3. Tạo và quản lý Index

3.1. Cú pháp

```
-- Index thường
CREATE INDEX ix_Accounts_status ON Accounts (status);

-- Composite (đẳng thức trước, khoảng sau)
CREATE INDEX ix_Orders_symbol_date ON Orders (symbol, order_date);

-- Unique
CREATE UNIQUE INDEX ux_Customers_phone ON Customers (phone);                          --
        MySQL
CREATE UNIQUE INDEX ux_Customers_phone ON Customers (phone) WHERE phone IS NOT NULL;  --
        SQL Server (cho nhiều NULL)

-- Xóa
DROP INDEX ix_Accounts_status ON Accounts;                                  -- cả SQL Server lẫn
        MySQL
```

Cách khác qua `ALTER TABLE`:

Việc	SQL Server	MySQL
Thêm index thường	(dùng `CREATE INDEX`)	`ALTER TABLE Accounts ADD INDEX ix_Accounts_status (status);`
Thêm ràng buộc duy nhất (kèm index)	`ALTER TABLE Customers ADD CONSTRAINT uq_x UNIQUE (col);`	`ALTER TABLE Customers ADD UNIQUE INDEX uq_x (col);`
Xóa	`ALTER TABLE Customers DROP CONSTRAINT uq_x;` hoặc `DROP INDEX`	`ALTER TABLE Accounts DROP INDEX ix_Accounts_status;`
Đổi cấu trúc index	`CREATE INDEX ... WITH (DROP_EXISTING = ON)`	Xóa rồi tạo lại

> **Mẹo:** đặt **tên tường minh** cho index (`ix_<Bảng>_<Cột>`, `ux_` cho unique) thay vì để hệ thống tự sinh: tên tự sinh như `PK__Orders__4659622983F810DE` chứa mã ngẫu nhiên, khác nhau giữa mỗi database, không dùng được trong script. Vì vậy các bài chấm ở Codelab đều yêu cầu **đặt đúng tên index** rồi kiểm tra theo tên.

Ví dụ tạo index rồi kiểm tra trong danh mục hệ thống (dùng bảng `MarketPrices` để không trùng bài tập):

SQL Server

```
CREATE INDEX ix_MarketPrices_date ON MarketPrices (trade_date);
SELECT i.name, i.type_desc, i.is_unique
FROM sys.indexes i
WHERE i.object_id = OBJECT_ID('MarketPrices') AND i.name LIKE 'ix[_]%';
```

MySQL

```
CREATE INDEX ix_MarketPrices_date ON MarketPrices (trade_date);
SELECT INDEX_NAME, COLUMN_NAME, SEQ_IN_INDEX, NON_UNIQUE
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'MarketPrices' AND LEFT(INDEX_NAME, 3) =
        'ix_';
```

Cả hai cho thấy index `ix_MarketPrices_date` (SQL Server: `NONCLUSTERED`, `is_unique = 0`; MySQL: `NON_UNIQUE = 1`).

Composite index: liệt kê từng cột theo thứ tự.

SQL Server

```
CREATE INDEX ix_Orders_symbol_date ON Orders (symbol, order_date);
SELECT i.name AS index_name, c.name AS column_name, ic.key_ordinal
FROM sys.indexes i
JOIN sys.index_columns ic ON ic.object_id = i.object_id AND ic.index_id = i.index_id
JOIN sys.columns c ON c.object_id = ic.object_id AND c.column_id = ic.column_id
WHERE i.object_id = OBJECT_ID('Orders') AND i.name = 'ix_Orders_symbol_date'
ORDER BY ic.key_ordinal;
```

MySQL

```
CREATE INDEX ix_Orders_symbol_date ON Orders (symbol, order_date);
SELECT INDEX_NAME AS index_name, COLUMN_NAME AS column_name, SEQ_IN_INDEX AS key_ordinal
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'Orders' AND INDEX_NAME =
        'ix_Orders_symbol_date'
ORDER BY SEQ_IN_INDEX;
```

Kết quả: `symbol` ở vị trí 1, `order_date` ở vị trí 2. Xóa index và kiểm tra lại:

```
CREATE INDEX ix_Securities_sector ON Securities (sector);
DROP INDEX ix_Securities_sector ON Securities;
SELECT COUNT(*) AS so_dong_securities FROM Securities;
```

Xóa index **không** ảnh hưởng dữ liệu bảng (số dòng vẫn nguyên).

3.2. Xem index đang có

SQL Server: trong SSMS, mở Object Explorer, chọn bảng, mở nhánh **Indexes**; hoặc truy vấn danh mục:

```sql
SELECT i.name, i.type_desc, i.is_unique, i.is_primary_key
FROM sys.indexes i
WHERE i.object_id = OBJECT_ID('Customers') ORDER BY i.name;
-- PK__Customer__CD65CB859651DCC5   CLUSTERED     1  1
-- UQ__Customer__6A9E4CB728092520   NONCLUSTERED  1  0     (từ ràng buộc UNIQUE của
        customer_code)
```

MySQL:

```sql
SHOW INDEX FROM Orders;
SELECT INDEX_NAME, COLUMN_NAME, SEQ_IN_INDEX, NON_UNIQUE
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = DATABASE() AND TABLE_NAME = 'Customers'
ORDER BY INDEX_NAME, SEQ_IN_INDEX;
-- customer_code  customer_code  1  0
-- PRIMARY        customer_id    1  0
-- referrer_id    referrer_id    1  1      (index tự tạo cho khóa ngoại)
```

> **Thực hành:** Vừa học xong cú pháp tạo index thường, composite, unique (kể cả khác biệt NULL giữa hai hệ) và cách kiểm tra trong danh mục hệ thống, hãy làm **bài 10.1**: tạo index trên khóa ngoại, composite `(account_id, order_date)`, unique index trên email cho phép nhiều NULL, và tạo/xóa index cho báo cáo giao dịch.

- **Bài 10.1** — Tạo index trên cột khóa ngoài *(độ khó: dễ)*
- **Bài 10.1** — Tạo composite index (account_id, order_date) *(độ khó: dễ)*
- **Bài 10.1** — Unique index trên email khách hàng *(độ khó: trung bình)*
- **Bài 10.1** — Tạo và xóa index cho báo cáo giao dịch *(độ khó: trung bình)*

3.3. Phân tích truy vấn: Execution Plan / EXPLAIN

Để biết database **thực sự** làm gì với truy vấn của bạn, hãy xem **kế hoạch thực thi**. Phần này không chấm tự động (kết quả phụ thuộc bản cài đặt), hãy làm theo từng bước trên bảng `BigTrades` ở mục 1.4.

SQL Server Management Studio (SSMS):

Bước	Thao tác	Bạn sẽ thấy
1	Mở cửa sổ truy vấn, gõ truy vấn cần phân tích	
2	Bấm **Query → Include Actual Execution Plan** (**Ctrl + M**). *Estimated plan* (Ctrl + L) không chạy truy vấn, chỉ ước lượng	
3	Chạy truy vấn (F5)	Cạnh tab *Results* xuất hiện tab **Execution Plan**
4	Đọc **từ phải sang trái**: dữ liệu chảy từ toán tử phải (đọc bảng) sang trái (kết quả)	Chuỗi hộp toán tử nối bằng mũi tên
5	Di chuột vào mũi tên để xem **Actual Rows**, vào toán tử xem **Estimated Cost**	Số dòng thực tế, chi phí
6	Bật `SET STATISTICS IO, TIME ON;` rồi xem tab *Messages*	Số trang đọc (logical reads), thời gian CPU

Toán tử	Ý nghĩa	Đánh giá
Table Scan / Clustered Index Scan	Đọc toàn bộ bảng	Cảnh báo nếu bảng lớn và chỉ cần vài dòng
Index Seek	Đi theo cây B-Tree tới đúng dòng	Tốt
Index Scan	Đọc toàn bộ index	Thường chấp nhận được, kém hơn Seek
Key Lookup (RID Lookup)	Tra ngược bảng để lấy cột thiếu	Nhiều lần thì cân nhắc covering index (`INCLUDE`)
Nested Loops / Hash Match / Merge Join	Cách JOIN	Hash Match trên bảng lớn có thể tốn bộ nhớ
Sort	Sắp xếp	Tốn; index có thứ tự phù hợp sẽ loại bỏ được

> **Lưu ý:** nếu thấy dấu **cảnh báo vàng**, hoặc SSMS gợi ý chữ xanh **Missing Index**, đó là manh mối tạo index. Nhưng đừng tạo mù quáng mọi gợi ý; hãy xét bảng có ghi nhiều không.

MySQL Workbench / dòng lệnh: đặt `EXPLAIN` trước câu `SELECT`.

```
EXPLAIN SELECT * FROM Orders WHERE account_id = 42 AND order_date >= '2026-09-01';
```

Trước khi có composite index, MySQL dùng index tự tạo cho khóa ngoại:

```
id  select_type  table   type  possible_keys  key         key_len  ref    rows  filtered
Extra
1   SIMPLE       Orders  ref   account_id     account_id  4        const  5     33.33
Using where
```

Sau `CREATE INDEX ix_Orders_account_date ON Orders (account_id, order_date)`:

```
1   SIMPLE      Orders  range ix_Orders_account_date  ix_Orders_account_date  7  NULL  3
100.0  Using index condition
```

Cột `EXPLAIN`	Ý nghĩa
`type`	Cách truy cập bảng, tốt → tệ: `const` / `eq_ref` → `ref` → `range` → `index` → **`ALL`** (quét toàn bảng)
`possible_keys` / `key`	Index có thể dùng / index thực sự được chọn (`NULL` = không dùng index)
`key_len`	Số byte index được dùng; với composite cho biết dùng được **mấy cột đầu**
`rows`	Số dòng ước tính phải đọc, càng nhỏ càng tốt
`filtered`	% dòng còn lại sau khi lọc
`Extra`	`Using index` (covering, rất tốt), `Using where` , `Using filesort` / `Using temporary` (tốn kém)

`EXPLAIN ANALYZE` (MySQL 8) chạy thật và in thời gian/số dòng thực tế, ví dụ `-> Index range scan on Trades using ix_Trades_trade_date ... (actual time=0.0215..0.024 rows=14 loops=1)` .

3.4. Index Hints / FORCE INDEX

Optimizer thường chọn kế hoạch tốt nhất dựa trên **thống kê** dữ liệu; đôi khi thống kê cũ hoặc truy vấn quá phức tạp khiến nó chọn sai. Khi đó bạn có thể **ép** dùng (hoặc tránh) một index:

Mục đích	MySQL	SQL Server
Gợi ý dùng index	`FROM BigTrades USE INDEX (ix_Big_date)`	`FROM BigTrades WITH (INDEX(ix_Big_date))`
Ép buộc dùng index	`FROM BigTrades FORCE INDEX (ix_Big_date)`	`WITH (FORCESEEK)` , `WITH (INDEX(ix_Big_date))`
Bỏ qua index	`IGNORE INDEX (ix_Big_date)`	`WITH (INDEX(0))` (ép quét bảng/clustered index)

Cú pháp hint chạy được ngay trên bảng mẫu:

SQL Server

```sql
CREATE INDEX ix_MarketPrices_date ON MarketPrices (trade_date);
GO
SELECT COUNT(*) AS so_dong
FROM MarketPrices WITH (INDEX(ix_MarketPrices_date))
WHERE trade_date = '2026-09-25';
```

MySQL

```sql
CREATE INDEX ix_MarketPrices_date ON MarketPrices (trade_date);
SELECT COUNT(*) AS so_dong
FROM MarketPrices FORCE INDEX (ix_MarketPrices_date)
WHERE trade_date = '2026-09-25';
```

Kết quả `12` (12 mã có giá ngày 2026-09-25). Hint không đổi kết quả, chỉ đổi cách tìm.

> **Cảnh báo:** chỉ dùng hint khi đã **đo** (execution plan + thời gian thật) thấy optimizer chọn sai và bạn không sửa được bằng cập nhật thống kê (`UPDATE STATISTICS` / `ANALYZE TABLE`) hoặc viết lại truy vấn. Hint là "băng dán": khi dữ liệu thay đổi, index bạn ép có thể trở thành lựa chọn tệ nhưng optimizer không còn quyền đổi. Nếu index bị xóa, truy vấn có hint sẽ **báo lỗi**.

3.5. Tối ưu truy vấn chậm: viết lại cho "sargable"

Nguyên nhân chậm phổ biến nhất không phải thiếu index mà là **hàm bọc quanh cột** trong `WHERE` : index trên cột đó bị bỏ qua vì database phải tính hàm cho từng dòng. Điều kiện dùng được index gọi là **sargable** (Search ARGument ABLE). Cách sửa: để **cột trần** ở một vế, đưa phép tính sang vế còn lại; với ngày, đổi `YEAR()/MONTH()` thành **khoảng ngày**.

Ví dụ trên bảng `Orders` : đếm lệnh tháng 09/2026.

```sql
-- Cách chậm: hàm bọc cột order_date
SELECT COUNT(*) AS so_lenh FROM Orders
WHERE YEAR(order_date) = 2026 AND MONTH(order_date) = 9;
```

```sql
-- Cách nhanh: khoảng ngày trên cột trần, cùng kết quả
SELECT COUNT(*) AS so_lenh FROM Orders
WHERE order_date >= '2026-09-01' AND order_date < '2026-10-01';
```

Hai truy vấn cho **cùng kết quả (72)**, nhưng chỉ truy vấn thứ hai tận dụng được index trên `order_date` . Dùng `< '2026-10-01'` thay vì `<= '2026-09-30'` để không bỏ sót dữ liệu khi cột có phần giờ.

Quy trình mỗi lần tối ưu:

Bước	Việc làm
1	Chạy truy vấn gốc, ghi lại số trang đọc / thời gian (mục 3.3)
2	Tìm điểm nghẽn: scan, key lookup, sort, hàm bọc cột
3	Viết lại truy vấn trước (sargable), rồi mới nghĩ tới thêm index
4	**Đo lại**, so sánh và ghi chú

> **Thực hành:** Vừa học xong cách đọc execution plan/ `EXPLAIN` và viết lại điều kiện thành sargable, hãy làm **bài 10.2**: chuyển truy vấn báo cáo giao dịch tháng 09/2026 dùng `YEAR()/MONTH()` thành điều kiện khoảng ngày, kết quả phải giữ nguyên. Sau khi nộp, thử tự bật execution plan / `EXPLAIN` cho hai phiên bản để thấy sự khác biệt.

- **Bài 10.2** — Tối ưu truy vấn: bỏ hàm trên cột ngày *(độ khó: trung bình)*

---

4. Transaction (giao tác) và toàn vẹn dữ liệu

Transaction là một nhóm câu lệnh được coi là **một đơn vị công việc không thể chia nhỏ**: hoặc **tất cả cùng thành công**, hoặc **không cái nào có hiệu lực**.

> **Ví dụ nghiệp vụ: chuyển 10.000.000 từ tài khoản A sang tài khoản B** gồm hai bước (trừ A, cộng B). Nếu hệ thống sập giữa hai bước mà không có transaction, tiền của khách hàng "bốc hơi".

4.1. Tính chất ACID

Chữ	Tính chất	Ý nghĩa	Ví dụ chuyển tiền
A	Atomicity (nguyên tử)	Tất cả hoặc không gì cả	Trừ A và cộng B cùng thành công, hoặc cả hai bị hủy
C	Consistency (nhất quán)	Dữ liệu luôn thỏa các ràng buộc/quy tắc	Tổng tiền hai tài khoản trước = sau; không vi phạm khóa ngoại, không âm
I	Isolation (cô lập)	Các transaction chạy đồng thời không làm hỏng nhau	Người khác không thấy trạng thái "đã trừ A nhưng chưa cộng B"
D	Durability (bền vững)	Đã `COMMIT` thì không mất, kể cả mất điện	Ghi vào transaction log trước khi báo thành công

4.2. Quản lý transaction

Lệnh	SQL Server	MySQL
Bắt đầu	`BEGIN TRANSACTION;`	`START TRANSACTION;` (hoặc `BEGIN;`)
Xác nhận	`COMMIT TRANSACTION;`	`COMMIT;`
Hoàn nguyên	`ROLLBACK TRANSACTION;`	`ROLLBACK;`
Điểm lưu	`SAVE TRANSACTION tên;`	`SAVEPOINT tên;`
Quay về điểm lưu	`ROLLBACK TRANSACTION tên;`	`ROLLBACK TO SAVEPOINT tên;`

Hình dưới nhắc lại (và mở rộng bằng SAVEPOINT) vòng đời của một transaction: mở bằng `BEGIN`, chạy các lệnh, rồi kết thúc bằng `COMMIT` (giữ mọi thay đổi) hoặc `ROLLBACK` (hủy mọi thay đổi).

Hình minh họa: Luồng của một transaction (xem hình động trên trang bài giảng trực tuyến).

Cách đọc hình: các lệnh nằm giữa `BEGIN` và điểm kết thúc chỉ là "nháp" đối với người khác cho tới khi `COMMIT`. `SAVEPOINT` là một mốc ở giữa: `ROLLBACK` về savepoint chỉ hủy phần **sau** mốc, phần trước mốc vẫn giữ và transaction tiếp tục.

> **Lưu ý:** cả hai hệ mặc định **autocommit**: mỗi câu lệnh đơn lẻ tự thành một transaction rồi tự `COMMIT`. `BEGIN/START TRANSACTION` tạm tắt chế độ này cho tới khi `COMMIT` / `ROLLBACK`.

Chuyển tiền có kiểm soát lỗi

SQL Server — bọc bằng `TRY ... CATCH`. Chuyển tiền sang tài khoản không tồn tại sẽ bị hoàn nguyên; câu cuối của khối `CATCH` cho thấy số dư tài khoản nguồn vẫn nguyên:

```
BEGIN TRY
    BEGIN TRANSACTION;
    UPDATE Accounts SET balance = balance - 5000000 WHERE account_id = 1;
    UPDATE Accounts SET balance = balance + 5000000 WHERE account_id = 99999;   -- tài khoản
        không tồn tại
    IF @@ROWCOUNT = 0 THROW 50001, 'Tai khoan dich khong ton tai', 1;
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
    SELECT ERROR_NUMBER() AS err_no, ERROR_MESSAGE() AS err_msg,
           (SELECT balance FROM Accounts WHERE account_id = 1) AS balance_acc1;
END CATCH;
```

Kết quả: `50001 Tai khoan dich khong ton tai 67000000.00`; khoản trừ 5.000.000 đã được hoàn nguyên.

MySQL — dùng `EXIT HANDLER` trong procedure:

```
DELIMITER //
CREATE PROCEDURE sp_TransferSafe(IN p_from INT, IN p_to INT, IN p_amount DECIMAL(18,2))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'ROLLED BACK' AS ket_qua,
               (SELECT balance FROM Accounts WHERE account_id = p_from) AS balance_from;
    END;
    START TRANSACTION;
    UPDATE Accounts SET balance = balance - p_amount WHERE account_id = p_from;
    UPDATE Accounts SET balance = balance + p_amount WHERE account_id = p_to;
    IF ROW_COUNT() = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tai khoan dich khong ton tai';
    END IF;
    COMMIT;
    SELECT 'COMMITTED' AS ket_qua, (SELECT balance FROM Accounts WHERE account_id = p_from)
        AS balance_from;
END //
DELIMITER ;
CALL sp_TransferSafe(1, 99999, 5000000);
```

Kết quả: `ROLLED BACK 67000000.00`.

Ba cách hoàn nguyên đúng:

Cách	Mô tả
1. Điều kiện ngay từ đầu	`... AND balance >= @amount` rồi kiểm tra số dòng bị ảnh hưởng (`@@ROWCOUNT` / `ROW_COUNT()`), không cần gây lỗi. Đây là cách của bài 10.3b
2. Gây lỗi có chủ đích	`THROW` / `SIGNAL` và để handler `ROLLBACK`
3. `XACT_ABORT ON` (T-SQL)	Tự rollback cả transaction khi có lỗi runtime

> **Cảnh báo:** các bẫy thường gặp: quên `COMMIT` (transaction "treo", giữ khóa, chặn người khác); `ROLLBACK` xong không thoát procedure nên các lệnh sau vẫn chạy ngoài transaction; DDL trong MySQL (`CREATE/ALTER/DROP`) **tự động commit**; MyISAM không hỗ trợ transaction (dùng InnoDB).

4.3. SAVEPOINT — hoàn nguyên một phần

Đặt "điểm lưu" giữa transaction; có thể quay về đó mà **vẫn giữ** phần trước và tiếp tục.

SQL Server

```sql
BEGIN TRANSACTION;
UPDATE Accounts SET balance = balance - 1000000 WHERE account_id = 1;
SAVE TRANSACTION sp_step1;
UPDATE Accounts SET balance = balance - 2000000 WHERE account_id = 1;
ROLLBACK TRANSACTION sp_step1;      -- chỉ hủy khoản trừ 2.000.000
COMMIT TRANSACTION;
SELECT balance FROM Accounts WHERE account_id = 1;
```

MySQL

```sql
START TRANSACTION;
UPDATE Accounts SET balance = balance - 1000000 WHERE account_id = 1;
SAVEPOINT sp_step1;
UPDATE Accounts SET balance = balance - 2000000 WHERE account_id = 1;
ROLLBACK TO SAVEPOINT sp_step1;     -- chỉ hủy khoản trừ 2.000.000
COMMIT;
SELECT balance FROM Accounts WHERE account_id = 1;
```

Cả hai cho `66000000.00`: chỉ còn khoản trừ 1.000.000.

Bước	Số dư tài khoản 1 (bắt đầu 67.000.000)
Sau `UPDATE` thứ nhất	66.000.000
Sau `SAVE TRANSACTION` / `SAVEPOINT`	66.000.000 (đánh dấu mốc)
Sau `UPDATE` thứ hai	64.000.000
Sau `ROLLBACK` về savepoint	66.000.000
Sau `COMMIT`	66.000.000 (đã ghi vĩnh viễn)

> **Lưu ý:** `ROLLBACK` không kèm tên savepoint trong T-SQL sẽ hủy **toàn bộ** transaction; savepoint không tự giải phóng khóa của phần đã hoàn nguyên.

> **Thực hành:** Vừa học xong `COMMIT`, `ROLLBACK`, `SAVEPOINT` và cách hoàn nguyên khi lỗi, hãy làm **bài 10.3**: chuyển tiền bằng transaction rồi hoàn nguyên thao tác nhầm, viết `sp_TransferMoney` an toàn (không làm gì khi thiếu tiền/tài khoản sai) và procedure hai khoản chi dùng SAVEPOINT. Phần isolation và deadlock ngay sau đó là hướng dẫn mô phỏng thủ công.

- **Bài 10.3** — Transaction: COMMIT chuyển tiền, ROLLBACK thao tác nhầm *(độ khó: dễ)*
- **Bài 10.3** — SP chuyển tiền giữa hai tài khoản (ROLLBACK khi lỗi) *(độ khó: trung bình)*
- **Bài 10.3** — SAVEPOINT: hoàn nguyên một phần transaction *(độ khó: khó)*

4.4. Isolation level và các xung đột đồng thời

Khi nhiều phiên (session) chạy cùng lúc trên cùng dữ liệu (nhiều nhân viên đặt lệnh, chạy báo cáo), có ba hiện tượng đọc bất thường:

Hiện tượng	Mô tả (ví dụ)
Dirty Read (đọc bẩn)	Phiên B đọc số dư account 1 vừa bị phiên A trừ nhưng **chưa COMMIT**; sau đó A `ROLLBACK` nên B đã đọc con số chưa từng tồn tại
Non-repeatable Read (đọc không lặp lại)	Trong cùng transaction, B đọc số dư account 1 = 67 triệu; A `COMMIT` sửa thành 57 triệu; B đọc lại thấy 57 triệu: **cùng một dòng, hai giá trị**
Phantom Read (đọc bóng ma)	B đếm lệnh của account 1 = 3 dòng; A `COMMIT` chèn thêm 1 lệnh; B đếm lại = 4: **xuất hiện dòng mới**

Isolation level quy định mức chống các hiện tượng trên (mức càng cao càng an toàn nhưng càng nhiều khóa, càng dễ chờ nhau/deadlock):

Mức	Dirty read	Non-repeatable read	Phantom read	Ghi chú
`READ UNCOMMITTED`	Có thể	Có thể	Có thể	Nhanh nhất, kém tin cậy (T-SQL: hint `NOLOCK`)
`READ COMMITTED`	Không	Có thể	Có thể	**Mặc định của SQL Server**
`REPEATABLE READ`	Không	Không	Có thể (theo chuẩn)	**Mặc định của MySQL InnoDB** (InnoDB dùng khóa khoảng nên phần lớn cũng chặn được phantom)
`SERIALIZABLE`	Không	Không	Không	Như chạy tuần tự, chậm và dễ chờ khóa
`SNAPSHOT` (chỉ SQL Server)	Không	Không	Không	Đọc từ "ảnh chụp" phiên bản dữ liệu, không chặn người ghi

```sql
-- Đổi mức cho phiên hiện tại (cú pháp giống nhau ở cả hai hệ)
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Thực hành mô phỏng dirty read (không chấm tự động vì cần hai phiên đồng thời). Mở **hai cửa sổ truy vấn** trong SSMS (hoặc hai tab trong Workbench), gọi là Phiên A và Phiên B, cùng kết nối tới `hnx_securities`:

Bước	Phiên A	Phiên B	Bạn sẽ thấy
1	`BEGIN TRANSACTION; UPDATE Accounts SET balance = balance - 1000000 WHERE account_id = 1;` (chưa commit)		A chạy xong, giữ khóa dòng
2		`SELECT balance FROM Accounts WHERE account_id = 1;`	Ở `READ COMMITTED` câu này **bị treo (chờ khóa)**
3		Hủy truy vấn treo, chạy `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;` rồi chạy lại	Đọc được số dư đã trừ (dirty read)
4	`ROLLBACK;`	Đọc lại	Số dư trở về giá trị cũ: B từng thấy một con số "ma"

> **Mẹo:** nhớ `ROLLBACK` ở phiên A khi xong để không giữ khóa; nếu phiên treo, đóng tab hoặc chạy `KILL <session_id>`.

4.5. Lock và Deadlock

Để giữ tính cô lập, database dùng **khóa (lock)**: khi A đang ghi một dòng, B muốn ghi/đọc dòng đó phải **chờ** tới khi A `COMMIT/ROLLBACK`. Chờ là bình thường.

Deadlock (khóa chết) xảy ra khi hai transaction **chờ lẫn nhau**, không bên nào tiến được. Cách mô phỏng: làm đúng bảng dưới trong hai cửa sổ truy vấn, thực hiện t1 ở cả hai phiên rồi t2 ở cả hai; trong vài giây một phiên nhận lỗi deadlock.

Thời điểm	Phiên A (chuyển tiền 1 → 2)	Phiên B (chuyển tiền 2 → 1)
t1	`BEGIN TRAN; UPDATE Accounts ... WHERE account_id = 1;` (giữ khóa dòng 1)	`BEGIN TRAN; UPDATE Accounts ... WHERE account_id = 2;` (giữ khóa dòng 2)
t2	`UPDATE Accounts ... WHERE account_id = 2;` → **chờ B**	`UPDATE Accounts ... WHERE account_id = 1;` → **chờ A**
t3	Cả hai chờ nhau mãi → deadlock. Database **chọn một nạn nhân** và hủy transaction của nó	

	SQL Server	MySQL InnoDB
Lỗi nạn nhân nhận	**1205** ("Transaction was deadlocked ... chosen as the deadlock victim")	**1213** ("Deadlock found when trying to get lock; try restarting transaction")
Sau đó	Transaction còn lại được chạy tiếp; nạn nhân phải `ROLLBACK` và thử lại	Tương tự

Cách phòng tránh:

1. **Luôn truy cập tài nguyên theo cùng một thứ tự**, ví dụ luôn khóa tài khoản có `account_id` nhỏ hơn trước (trong `sp_TransferMoney`, `UPDATE` theo thứ tự id tăng dần bất kể chiều chuyển). Đây là biện pháp hiệu quả nhất.
2. **Giữ transaction ngắn:** chuẩn bị dữ liệu và tính toán **trước** khi `BEGIN`, không chờ người dùng nhập liệu khi transaction đang mở.
3. **Có index phù hợp** để câu `UPDATE/DELETE` chỉ khóa đúng dòng cần (không có index thì khóa nhiều dòng/quét cả bảng).
4. Chọn isolation level vừa đủ (đừng dùng `SERIALIZABLE` nếu không cần); SQL Server cân nhắc `READ_COMMITTED_SNAPSHOT`.
5. **Cài cơ chế thử lại (retry)** ở ứng dụng khi bắt lỗi 1205/1213.

Theo dõi: SQL Server dùng Activity Monitor, `sp_who2`, `sys.dm_tran_locks`, Extended Events `xml_deadlock_report` (phiên `system_health` có sẵn); MySQL dùng `SHOW ENGINE INNODB STATUS` (mục *LATEST DETECTED DEADLOCK*), `performance_schema.data_locks`.

---

5. Hiệu năng (Performance Tuning)

5.1. Viết truy vấn hiệu quả

1. **Chỉ lấy thứ cần:** liệt kê cột thay vì `SELECT *`; lọc sớm bằng `WHERE`; phân trang bằng `OFFSET ... FETCH` (T-SQL) / `LIMIT` (MySQL) thay vì kéo cả bảng về ứng dụng.
2. **Điều kiện "sargable"** (đã làm ở bài 10.2): để **cột trần ở một vế**, đưa phép tính sang vế còn lại.

Không sargable (index bị bỏ qua)	Sargable
`WHERE YEAR(trade_date) = 2026 AND MONTH(trade_date) = 9`	`WHERE trade_date >= '2026-09-01' AND trade_date < '2026-10-01'`
`WHERE quantity * 2 > 1000`	`WHERE quantity > 500`
`WHERE LEFT(full_name, 4) = 'Ngan'`	`WHERE full_name LIKE 'Ngan%'`
`WHERE full_name LIKE '%Hang'` (dấu `%` ở đầu)	full-text index hoặc thiết kế lại
`WHERE ISNULL(city, '') = 'Ha Noi'`	`WHERE city = 'Ha Noi'`

3. **Tránh chuyển kiểu ngầm:** so sánh cột chuỗi với số (`WHERE customer_code = 100007`) khiến database chuyển đổi cả cột và mất index. So sánh đúng kiểu dữ liệu.
4. **Dùng JOIN/EXISTS thay cho subquery tương quan** chạy lại cho từng dòng; dùng `EXISTS` khi chỉ cần biết "có hay không".
5. **Xử lý theo tập hợp**, không theo từng dòng (vòng lặp/cursor). Một `UPDATE` tập hợp thường nhanh hơn vòng lặp hàng trăm lần.

5.2. Common Mistakes (lỗi hiệu năng thường gặp)

Lỗi	Hậu quả	Cách sửa
`SELECT *` mọi nơi	Đọc thừa dữ liệu, phá covering index	Liệt kê cột
Hàm bọc cột trong `WHERE`	Không dùng được index	Viết lại dạng khoảng
Thiếu index cho khóa ngoại dùng `JOIN` (SQL Server)	JOIN quét toàn bảng	`CREATE INDEX` trên khóa ngoại
`UPDATE` / `DELETE` thiếu `WHERE`	Sửa/xóa cả bảng	Luôn thử bằng `SELECT` trước; dùng transaction
Transaction dài / quên `COMMIT`	Chặn người khác, deadlock	Ngắn gọn, luôn có `COMMIT/ROLLBACK`
`NOT IN (subquery)` gặp `NULL`	Trả về rỗng bất ngờ	Dùng `NOT EXISTS`

5.3. Monitoring & Optimization

Việc	SQL Server	MySQL
Xem kế hoạch thực thi	SSMS: *Include Actual Execution Plan*	`EXPLAIN` / `EXPLAIN ANALYZE`
Đo số trang đọc / thời gian	`SET STATISTICS IO, TIME ON;`	`EXPLAIN ANALYZE` , `SHOW PROFILE`
Tìm truy vấn chậm	**Query Store**, `sys.dm_exec_query_stats` , Activity Monitor	**Slow Query Log**, `performance_schema`
Xem phiên đang chạy	`sp_who2` , `sys.dm_exec_requests`	`SHOW PROCESSLIST`
Gợi ý index còn thiếu	`sys.dm_db_missing_index_details`	(Workbench Performance Reports)
Index ít dùng (đáng xóa)	`sys.dm_db_index_usage_stats`	`sys.schema_unused_indexes`
Bảo trì index bị phân mảnh	`ALTER INDEX ... REORGANIZE / REBUILD`	`OPTIMIZE TABLE`
Cập nhật thống kê	`UPDATE STATISTICS Trades;`	`ANALYZE TABLE Trades;`

> **Mẹo:** quy trình tối ưu bài bản: (1) xác định truy vấn chậm bằng số liệu thật, (2) xem execution plan tìm điểm nghẽn (scan, key lookup, sort), (3) sửa: viết lại truy vấn trước, rồi mới thêm index, (4) **đo lại** và so sánh, (5) ghi chú lại. Tối ưu "theo cảm giác" thường phản tác dụng.

---

6. Kết nối và trích xuất dữ liệu SQL trực tiếp từ Excel

Nhân viên phân tích thường cần kéo dữ liệu từ database vào Excel để lập bảng tổng hợp/biểu đồ, và muốn **bấm Refresh là số liệu tự cập nhật** thay vì copy-paste. Codelab không chấm phần này; bạn làm theo các bước dưới

đây trên máy tính công ty.

6.1. Tổng quan các phương thức kết nối

Phương thức	Dùng khi	Ghi chú
Get Data → From SQL Server Database (connector gốc)	Kết nối SQL Server	Nhanh, ổn định, hỗ trợ Windows Authentication
Get Data → From Other Sources → From ODBC	Kết nối MySQL và hầu hết CSDL khác	Cần cài **ODBC driver** (vd. *MySQL Connector/ODBC*) và tạo DSN
Get Data → From MySQL Database	MySQL (trên bản Excel có connector)	Cần cài MySQL Connector/NET
Microsoft Query (cách cũ)	Excel đời cũ	Vẫn dùng được, kém linh hoạt hơn Power Query

Cả hai đường đi qua **Power Query** (công cụ nạp và biến đổi dữ liệu trong Excel 2016+/Microsoft 365, nằm trong tab **Data**).

6.2. Các bước: kết nối SQL Server

Bước	Thao tác	Bạn sẽ thấy
1	Mở Excel, tab **Data** → **Get Data** → **From Database** → **From SQL Server Database**	Hộp thoại nhập server
2	Nhập **Server** (tên máy chủ hoặc `tenmay\INSTANCE`) và **Database** (`hnx_securities`)	
3	Mở **Advanced options** nếu muốn dán một câu SQL vào ô **SQL statement** (khuyến khích), ví dụ `SELECT customer_id, full_name, status FROM Customers WHERE city = 'Ha Noi'`, rồi bấm **OK**	
4	Chọn xác thực: **Windows** (ưu tiên, không phải nhập mật khẩu) hoặc **Database** (tên đăng nhập/mật khẩu SQL), bấm **Connect**. Nếu hiện cảnh báo mã hóa, chọn theo chính sách công ty	Cửa sổ **Navigator**
5	Tick bảng/view cần lấy. Nếu dùng câu SQL riêng thì kết quả hiện luôn ở bước xem trước	Bảng xem trước bên phải
6	Chọn **Load** (nạp thẳng vào sheet dạng Table) hoặc **Transform Data** (mở **Power Query Editor** để lọc cột, đổi kiểu, gộp bảng, thêm cột tính, rồi **Close & Load**)	Dữ liệu xuất hiện trong sheet

6.3. Các bước: kết nối MySQL (hoặc CSDL khác) bằng ODBC

Bước	Thao tác
1	Cài **MySQL Connector/ODBC** (bản 64-bit khớp với Excel 64-bit)
2	Mở **ODBC Data Sources (64-bit)** của Windows, tab *User DSN* → **Add** → chọn *MySQL ODBC Unicode Driver* → nhập tên DSN (vd. `hnx_mysql`), Server, Port 3306, User, Password, Database `hnx_securities` → **Test** → **OK**
3	Trong Excel: **Data** → **Get Data** → **From Other Sources** → **From ODBC**
4	Chọn DSN vừa tạo; có thể dán câu SQL vào **Advanced options** → **SQL statement**; nhập thông tin đăng nhập (tab *Default or Custom*/*Database*) → **Connect**
5	Chọn bảng, rồi **Load** hoặc **Transform Data** như trên

Trong Power Query Editor có thể xem công thức M sinh ra, ví dụ:

```
= Sql.Database("tenmay", "hnx_securities", [Query = "SELECT customer_id, full_name FROM Customers"])
```

6.4. Thao tác trích xuất và xử lý

- **Lọc / chọn cột / đổi kiểu** ngay trong Power Query (mỗi thao tác là một bước ghi lại, có thể sửa/xóa).
- **Làm mới dữ liệu:** tab **Data** → **Refresh All** (Ctrl + Alt + F5). Đặt tự động: **Data** → **Queries & Connections** → chuột phải truy vấn → **Properties** → *Refresh every N minutes* hoặc *Refresh data when opening the file*.
- **Dùng dữ liệu:** chèn **PivotTable/PivotChart** trên bảng đã nạp; dùng `SUMIFS`, `XLOOKUP`, Slicer; hoặc tạo **Connection only** rồi `Load To → PivotTable Report`.
- **Xuất kết quả:** bảng đã nạp lưu ngay trong file Excel (hoặc xuất CSV).

> **Mẹo:** đẩy phần **tính toán nặng vào câu SQL** (`GROUP BY`, `SUM`) để Excel chỉ nhận vài trăm dòng tổng hợp thay vì hàng triệu dòng chi tiết.

6.5. Ứng dụng thực tế và lưu ý an toàn

Nhu cầu	Cách làm
Báo cáo tài sản khách hàng hằng ngày	Dán truy vấn "Top 10 khách hàng theo tài sản" (chính là bài 10.4a) vào Power Query → PivotChart → mỗi sáng bấm Refresh
Theo dõi giá trị giao dịch theo tháng	Kéo kết quả `sp_MonthlyReport` hoặc truy vấn 10.4c vào Excel để vẽ biểu đồ đường
Báo cáo dùng lại nhiều nơi	Nhờ đội IT chuẩn hóa thành view/stored procedure (có index tối ưu) rồi Excel chỉ gọi tên

> **Cảnh báo:** dùng **tài khoản chỉ đọc** (chỉ `SELECT`) cho Excel; ưu tiên Windows Authentication; **không lưu mật khẩu** trong file chia sẻ; dữ liệu khách hàng là dữ liệu nhạy cảm, không gửi file Excel chứa dữ liệu thật ra ngoài công ty; thu hẹp phạm vi dữ liệu bằng `WHERE`; tránh `SELECT *` trên bảng lớn.

7. Review và hệ thống hóa toàn bộ khóa học

7.1. Bản đồ 10 buổi

Buổi	Chủ đề	Từ khóa cần nhớ
1	Giới thiệu SQL, SELECT cơ bản	`SELECT`, `FROM`, `WHERE`, `AND/OR`, `IN`, `BETWEEN`, `LIKE`, `DISTINCT`
2	Sắp xếp và hàm thống kê	`ORDER BY`, `COUNT/SUM/AVG/MIN/MAX`, `GROUP BY`, `HAVING`
3	Window Functions	`OVER (PARTITION BY ... ORDER BY ...)`, `ROW_NUMBER`, `RANK`, `LAG/LEAD`, tổng lũy kế
4	Kiểu dữ liệu và hàm xử lý	`CAST/CONVERT`, hàm chuỗi, hàm ngày, `CASE`, `COALESCE`
5	CTE, Temp Table, Subquery	`WITH ... AS`, bảng tạm, subquery vô hướng / nhiều dòng / tương quan, `EXISTS`
6	INSERT, UPDATE, DELETE	Thay đổi dữ liệu, `WHERE` an toàn, transaction
7	Quan hệ giữa bảng, Normal Form	Khóa chính/ngoại, 1-1 / 1-N / N-N, 1NF–3NF
8	JOIN	`INNER`, `LEFT`, `RIGHT`, `FULL`, `SELF JOIN`
9	Stored Procedure và Trigger	`CREATE PROCEDURE`, tham số, `IF/WHILE`, `AFTER` trigger, `inserted/deleted` / `NEW/OLD`
10	Index, Transaction, Review	B-Tree, composite, ACID, `COMMIT/ROLLBACK/SAVEPOINT`, deadlock

7.2. Sơ đồ ER hệ thống chứng khoán (nhắc lại)

Bộ dữ liệu xuyên suốt khóa gồm 9 bảng; sơ đồ dưới nhắc lại các quan hệ chính để bạn xác định đường JOIN cho Capstone.

Hình minh họa: Sơ đồ ER của bộ dữ liệu mẫu (xem hình động trên trang bài giảng trực tuyến).

Đường đi từ giao dịch tới khách hàng: `Trades → Orders → Accounts → Customers`. Thống kê nằm ở `CustomerStats` (1-1 với Customers), nhật ký ở `AuditLog` (1-N với Customers).

7.3. Danh sách hàm thường dùng (cheat sheet SQL Server ↔ MySQL)

Nhóm	SQL Server	MySQL
Ghép chuỗi	`CONCAT`, `+`	`CONCAT`
Độ dài chuỗi	`LEN` (bỏ khoảng trắng cuối)	`CHAR_LENGTH`
Tìm vị trí	`CHARINDEX(x, s)`	`INSTR(s, x)` / `LOCATE(x, s)`
Chuỗi con	`SUBSTRING`, `LEFT`, `RIGHT`	`SUBSTRING`, `LEFT`, `RIGHT`
Ngày: năm/tháng/ngày	`YEAR`, `MONTH`, `DAY`	`YEAR`, `MONTH`, `DAY`
Cộng ngày	`DATEADD(DAY, n, d)`	`DATE_ADD(d, INTERVAL n DAY)`
Khoảng cách ngày	`DATEDIFF(DAY, a, b)`	`DATEDIFF(b, a)`
Định dạng ngày	`FORMAT(d, 'yyyy-MM')`	`DATE_FORMAT(d, '%Y-%m')`
Xử lý NULL	`ISNULL`, `COALESCE`	`IFNULL`, `COALESCE`
Điều kiện	`CASE`, `IIF`	`CASE`, `IF()`
Gộp chuỗi nhóm	`STRING_AGG`	`GROUP_CONCAT`
Lấy N dòng	`TOP n` / `OFFSET..FETCH`	`LIMIT n [OFFSET m]`
Bảng tạm	`#tmp`, `SELECT ... INTO #tmp`	`CREATE TEMPORARY TABLE`
Làm tròn	`ROUND`	`ROUND`

7.4. Best Practices và Anti-patterns (tổng kết)

Nên	Tránh (anti-patterns)
Đặt tên rõ ràng, nhất quán; viết hoa từ khóa, thụt lề	`SELECT *`; hàm bọc cột trong `WHERE`; `NOT IN` với cột có `NULL`
Luôn có `ORDER BY` khi cần thứ tự ổn định, thêm khóa phụ khi có thể đồng hạng	`UPDATE/DELETE` không `WHERE`; sửa trực tiếp trên production không backup
Liệt kê cột; lọc sớm; xử lý theo tập hợp; dùng CTE cho truy vấn dài	Vòng lặp/cursor thay câu lệnh tập hợp; dồn mọi logic nghiệp vụ vào trigger
Bọc thay đổi dữ liệu trong transaction; thử bằng `SELECT` trước khi `UPDATE/DELETE`	Tạo index tràn lan; transaction dài; quên `COMMIT`
Bọc `CAST(ROUND(x, 2) AS DECIMAL(18,2))` khi báo cáo tỷ lệ; nhớ chia số nguyên trên SQL Server	Chia hai số nguyên rồi tin kết quả thập phân
Xử lý `NULL` chủ động (`COALESCE`, `IS NULL`); index cho khóa ngoại/cột lọc, đo trước-sau	Tối ưu theo cảm giác không đo

7.5. Capstone Project — báo cáo phân tích khách hàng (bài 10.4)

Mini project cho phòng chăm sóc khách hàng. Bạn sẽ ráp các mảnh của cả khóa thành một bộ báo cáo, giống công việc thực tế. Đây là phần thực hành lớn nhất của buổi, làm **sau cùng**.

Bài	Nội dung	Kỹ năng ôn lại
10.4a	Top 10 khách hàng theo tài sản	JOIN, `GROUP BY`, CTE, window `ROW_NUMBER`, `TOP` / `LIMIT`
10.4b	Tổng giao dịch theo **ngày** (tháng 09/2026)	`GROUP BY`, điều kiện khoảng ngày (sargable)
10.4c	Tổng giao dịch theo **tháng**, lũy kế trong năm	CTE, hàm ngày, window `SUM() OVER (PARTITION BY ...)`
10.4d	Tổng giao dịch theo **năm** và tỷ trọng	CTE, `SUM() OVER ()`, `CAST/ROUND`
10.4e	Phân loại khách hàng **VIP / Thuong / Moi**	`LEFT JOIN`, `CASE`, `COALESCE`
10.4f	Top 3 khách hàng giao dịch nhiều nhất mỗi loại	nhiều `JOIN`, CTE, `ROW_NUMBER() OVER (PARTITION BY ...)`
10.4g	Stored Procedure báo cáo theo tháng	Procedure, tham số, hàm ngày, sargable
10.4h	Hồ sơ 360 độ khách hàng	Tổng hợp mọi kỹ thuật

Cách tiếp cận đề xuất:

1. Đọc đề, xác định **bảng và đường JOIN** (`Trades → Orders → Accounts → Customers`).
2. Viết **từng CTE nhỏ** và kiểm tra kết quả từng bước.
3. Ghép lại; kiểm tra trường hợp biên (khách hàng không có tài khoản/giao dịch, đồng hạng, `NULL`).
4. Đóng gói truy vấn dùng lại thành **Stored Procedure** (bài 10.4g) và thêm **index** phục vụ báo cáo (như bài 10.1), rồi xem `EXPLAIN`/execution plan.

Mẫu ghép báo cáo (dùng chủ đề khác để bạn thấy khung xương: CTE tổng hợp, `ROW_NUMBER` xếp hạng, lọc top N). Top 5 mã chứng khoán theo giá trị giao dịch:

```sql
WITH v AS (
    SELECT symbol, COUNT(*) AS trade_count, SUM(quantity * price) AS total_value
    FROM Trades
    GROUP BY symbol
), r AS (
    SELECT symbol, trade_count, total_value,
           ROW_NUMBER() OVER (ORDER BY total_value DESC, symbol) AS rk
    FROM v
)
SELECT rk, symbol, trade_count, total_value
FROM r
WHERE rk <= 5
ORDER BY rk;
```

> **Lưu ý:** hai quy tắc phân loại dùng thống nhất trong 10.4e và 10.4h: `VIP` nếu tổng tài sản ACTIVE từ 1 tỷ trở lên; `Moi` nếu `created_date >= '2026-06-27'` (90 ngày tính đến 2026-09-25); còn lại `Thuong`. Kiểm tra theo thứ tự từ trên xuống: một khách hàng vừa đủ điều kiện VIP vừa mới mở tài khoản vẫn là VIP.

> **Thực hành:** Vừa ôn xong toàn bộ khóa, hãy làm **bài 10.4 (Capstone)** theo thứ tự a → h: từ Top 10 tài sản, báo cáo giao dịch theo ngày/tháng/năm, phân loại khách hàng, top 3 mỗi loại, procedure báo cáo cho tới hồ sơ 360 độ. Mỗi bài là một mảnh của báo cáo hoàn chỉnh; làm lần lượt và kiểm tra kỹ trường hợp biên trước khi nộp.

- **Bài 10.4** — Capstone: Top 10 khách hàng theo tài sản *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo ngày (tháng 09/2026) *(độ khó: dễ)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo tháng kèm lũy kế năm *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo năm và tỷ trọng *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Phân loại khách hàng VIP / Thuong / Moi *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Top 3 khách hàng giao dịch nhiều nhất mỗi loại *(độ khó: khó)*
- **Bài 10.4** — Capstone: SP báo cáo giao dịch theo tháng *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Hồ sơ 360 độ khách hàng có giao dịch *(độ khó: khó)*

Sai lầm thường gặp

Sai lầm	Hậu quả	Cách đúng
Tạo index trên bảng nhỏ hoặc cột ít giá trị (`status` , `side`)	Optimizer bỏ qua, chỉ tốn dung lượng và làm chậm ghi	Index cột chọn lọc cao, dùng trong `WHERE` / `JOIN`
Tạo composite với thứ tự cột sai (khoảng trước, đẳng thức sau)	Không tận dụng hết index	Cột `=` trước, cột khoảng sau; nhớ quy tắc cột trái nhất
Không đặt tên index tường minh	Tên tự sinh khác nhau mỗi database, script không tái dùng được	`ix_<Bảng>_<Cột>` , `ux_...`
Unique index trên cột cho phép NULL (SQL Server)	Lỗi vì dữ liệu đã có nhiều NULL	Filtered index `WHERE col IS NOT NULL`
Bọc hàm quanh cột (`YEAR(trade_date)`)	Index không được dùng, quét cả bảng	Viết lại thành khoảng ngày
Quên `COMMIT` / `ROLLBACK`	Transaction treo, giữ khóa, chặn người khác	Luôn kết thúc transaction, giữ ngắn
Cho phép thao tác âm số dư vì không kiểm tra `@@ROWCOUNT` / `ROW_COUNT()`	Cộng tiền vào tài khoản không tồn tại, tiền "biến mất"	Kiểm tra số dòng ảnh hưởng sau từng `UPDATE` và `ROLLBACK` khi 0
Hai phiên khóa tài nguyên theo thứ tự ngược nhau	Deadlock	Luôn `UPDATE` theo thứ tự id tăng dần, có retry
Ép index bằng hint rồi quên	Index bị xóa thì truy vấn báo lỗi; dữ liệu đổi thì kế hoạch tệ	Chỉ hint khi đã đo; ghi chú lý do
Dán mật khẩu vào file Excel chia sẻ	Lộ dữ liệu khách hàng	Tài khoản chỉ đọc, Windows Authentication

Tóm tắt

Chủ đề	Ghi nhớ
Index (B-Tree)	Tăng tốc đọc, làm chậm ghi và tốn dung lượng; index chỉ được dùng khi có lợi
Loại index	PK (clustered), UNIQUE, composite (cột trái nhất, `=` trước khoảng sau), covering (`INCLUDE`), FULLTEXT
Khóa ngoại	SQL Server không tự tạo index cho khóa ngoại; MySQL InnoDB tự tạo
Unique + NULL	SQL Server chỉ cho 1 NULL → dùng filtered index; MySQL cho nhiều NULL
Quản lý index	`CREATE INDEX` , `DROP INDEX ... ON ...` , đặt tên tường minh
Phân tích	SSMS *Include Actual Execution Plan*; MySQL `EXPLAIN` (tránh `type = ALL`); hint/ `FORCE INDEX` chỉ khi đã đo
Transaction	ACID; `BEGIN/START TRANSACTION` , `COMMIT` , `ROLLBACK` , `SAVEPOINT`
Isolation	Dirty / Non-repeatable / Phantom read; mặc định SQL Server = READ COMMITTED, MySQL = REPEATABLE READ
Deadlock	Hai phiên chờ nhau; phòng tránh: cùng thứ tự truy cập, transaction ngắn, có index, retry
Hiệu năng	Truy vấn sargable, không `SELECT *` , xử lý theo tập hợp, đo trước-sau
Excel	Data → Get Data → From Database (SQL Server) / From ODBC; Load hoặc Transform; Refresh All
Capstone	CTE nhỏ ghép lại, kiểm tra biên, đóng gói procedure, thêm index

Gợi ý mở rộng

- Thử tạo index cho từng truy vấn 10.4 rồi so sánh execution plan trước/sau trên bảng `BigTrades` ; ghi lại số trang đọc.
- Viết procedure `sp_TransferMoney` phiên bản **chống deadlock** (luôn `UPDATE` tài khoản có id nhỏ hơn trước).
- Tìm hiểu **View** và **indexed view** (SQL Server) cho báo cáo dùng lại nhiều lần; **partitioning** cho bảng giao dịch rất lớn.
- Tìm hiểu bảo mật ở mức database: phân quyền `GRANT EXECUTE` trên procedure, tài khoản chỉ đọc cho Excel/BI.
- Tự xây dựng **bộ SQL snippet** cá nhân từ các bài của khóa để dùng lại trong công việc hằng ngày.

Bài tập thực hành của buổi

- **Bài 10.1** — Tạo index trên cột khóa ngoài *(độ khó: dễ)*
- **Bài 10.1** — Tạo composite index (account_id, order_date) *(độ khó: dễ)*
- **Bài 10.1** — Unique index trên email khách hàng *(độ khó: trung bình)*
- **Bài 10.1** — Tạo và xóa index cho báo cáo giao dịch *(độ khó: trung bình)*
- **Bài 10.2** — Tối ưu truy vấn: bỏ hàm trên cột ngày *(độ khó: trung bình)*
- **Bài 10.3** — Transaction: COMMIT chuyển tiền, ROLLBACK thao tác nhầm *(độ khó: dễ)*

- **Bài 10.3** — SP chuyển tiền giữa hai tài khoản (ROLLBACK khi lỗi) *(độ khó: trung bình)*
- **Bài 10.3** — SAVEPOINT: hoàn nguyên một phần transaction *(độ khó: khó)*
- **Bài 10.4** — Capstone: Top 10 khách hàng theo tài sản *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo ngày (tháng 09/2026) *(độ khó: dễ)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo tháng kèm lũy kế năm *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Tổng giao dịch theo năm và tỷ trọng *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Phân loại khách hàng VIP / Thuong / Moi *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Top 3 khách hàng giao dịch nhiều nhất mỗi loại *(độ khó: khó)*
- **Bài 10.4** — Capstone: SP báo cáo giao dịch theo tháng *(độ khó: trung bình)*
- **Bài 10.4** — Capstone: Hồ sơ 360 độ khách hàng có giao dịch *(độ khó: khó)*

Nộp bài tại trang bài tập của khóa trên Codelab.

Thực hành trên Windows: đo hiệu năng index, quan sát khóa và nối Excel

Các bước dưới đây dùng bảng lớn `BigTrades` ở mục 1.4. Nếu chưa tạo, hãy chạy lại các lệnh của mục đó trước.

1. Đo trước và sau khi tạo index.

1. Nhấn **Ctrl+M** (*Include Actual Execution Plan*) và chạy `SET STATISTICS IO ON; SET STATISTICS TIME ON;`.
2. Chạy truy vấn lọc theo ngày khi **chưa** có index. Ở tab **Execution Plan** bạn sẽ thấy *Table Scan* (hoặc *Clustered Index Scan*), ở tab **Messages** ghi *logical reads* lớn.
3. Tạo index theo mục 1.4 rồi chạy lại: kế hoạch chuyển sang **Index Seek**, *logical reads* giảm mạnh.
4. Ghi hai con số *logical reads* vào ghi chú của bạn để so sánh.

Nếu SSMS hiện dòng chữ xanh **Missing Index (Impact ...)** ở đầu kế hoạch, chuột phải → **Missing Index Details...** để xem câu lệnh `CREATE INDEX` gợi ý; chỉ lấy làm tham khảo.

2. Xem index qua giao diện. Object Explorer: bảng → **Indexes**; chuột phải một index → **Properties** (tab *General*, *Fragmentation*). Xem danh sách bằng lệnh:

```
SELECT name AS ten_index, type_desc, is_unique
FROM sys.indexes
WHERE object_id = OBJECT_ID('Accounts') AND name IS NOT NULL;
```

3. Quan sát khóa (lock) giữa hai cửa sổ. Mở hai cửa sổ truy vấn cùng kết nối `hnx_securities`:

Bước	Cửa sổ A	Cửa sổ B
1	`BEGIN TRANSACTION; UPDATE Accounts SET balance = balance + 1 WHERE account_id = 1;`	
2		`SELECT balance FROM Accounts WHERE account_id = 1;` → **treo chờ**
3	Mở **Activity Monitor** (**Ctrl+Alt+A**) hoặc chạy ở cửa sổ thứ ba: `SELECT session_id, blocking_session_id, wait_type FROM sys.dm_exec_requests WHERE blocking_session_id <> 0;`	
4	`ROLLBACK;`	Câu truy vấn chạy tiếp và trả về giá trị ban đầu

> **Cảnh báo:** luôn `ROLLBACK` hoặc `COMMIT` ở cửa sổ A sau khi thử. Nếu cửa sổ bị treo, bấm nút **Cancel Executing Query** (hình vuông đỏ) rồi chạy `ROLLBACK` .

4. Nối Excel với SQL Server (Windows). Mục 6 trình bày chi tiết; thao tác nhanh trong Excel: **Data** → **Get Data** → **From Database** → **From SQL Server Database**, nhập Server `localhost\SQLEXPRESS` , Database `hnx_securities` , ở mục *Advanced options* dán câu `SELECT` của bạn, **OK**, chọn *Windows* và **Connect**, cuối cùng **Load**. Muốn cập nhật số liệu: **Data** → **Refresh All**.

5. Gói báo cáo cuối khóa (Capstone). Lưu từng câu truy vấn thành file `.sql` đặt tên rõ ràng (`01_top10_khach_hang.sql` ...), tạo procedure và index theo yêu cầu bài 10.4, xuất kết quả sang Excel theo bước 4 hoặc **Save Results As...** (CSV), rồi chụp màn hình kế hoạch thực thi trước/sau index để đưa vào báo cáo.

Với MySQL Workbench: dùng `EXPLAIN` hoặc nút **Visual Explain**; xem khóa ở `SHOW ENGINE INNODB STATUS;` ; Excel kết nối qua *MySQL for Excel* hoặc ODBC.

Tự kiểm tra: hoàn thành bảng so sánh "chưa index / có index" (loại toán tử, logical reads, thời gian) cho truy vấn của bạn, rồi nộp các bài 10.1–10.4 trên Codelab.