Tổng hợp Kiến thức Môn Kỹ thuật Dữ liệu (Data Engineering)

Học viên Cao học KHMT Bách Khoa TP.HCM (HCMUT)

Ngày 4 tháng 1 năm 2026

1 Nguyên lý Phân tích & Thiết kế CSDL

1.1 Tổng quan Các giai đoạn Thiết kế

1. **Mức Quan niệm (Conceptual):** *Mục tiêu* — nắm bắt yêu cầu và ngữ nghĩa (độc lập với cài đặt); *Mô hình/Công cụ* — ER/EER; *Đầu ra* — lược đồ quan niệm (thực thể, thuộc tính, mối kết hợp, ràng buộc).
2. **Mức Logic:** *Mục tiêu* — ánh xạ từ mức quan niệm sang mô hình DBMS đích (ví dụ: quan hệ); *Mô hình/Công cụ* — ánh xạ ER-sang-quan hệ, chuẩn hóa (FDs); *Đầu ra* — lược đồ quan hệ (bảng, khóa, ràng buộc toàn vẹn).
3. **Mức Vật lý:** *Mục tiêu* — xác định cấu trúc lưu trữ và đường dẫn truy xuất để tối ưu hiệu năng; *Mô hình/Công cụ* — phân tích tải, chỉ mục, tổ chức tập tin, băm; *Đầu ra* — lược đồ trong (cấu trúc lưu trữ, chỉ mục, đường dẫn truy xuất).

1.2 Nguyên lý Thiết kế Mức Quan niệm

- **Phân tích Yêu cầu:** Làm việc với người dùng/chuyên gia nghiệp vụ để nắm bắt yêu cầu dữ liệu và yêu cầu chức năng (thao tác/giao dịch).
- **Thành phần ER: Thực thể** (vd: NhanVien), **Thuộc tính** (đơn/phức hợp/đa trị/dẫn xuất), **Mối kết hợp** (sự liên kết giữa các thực thể).
- **Ràng buộc Cấu trúc:** Tỷ số bản số (1:1, 1:N, M:N) và ràng buộc tham gia (*toàn phần* vs *từng phần*).
- **Thực thể Yếu:** Được xác định thông qua **mối kết hợp xác định** với một thực thể **chủ** và một **khóa bộ phận**; thực thể yếu tham gia *toàn phần* vào mối kết hợp xác định.
- **Tinh chỉnh Top-Down:** Tinh chỉnh lặp lại các thực thể tổng quát; áp dụng chuyên biệt hóa/tổng quát hóa (EER).

1.3 Nguyên lý Thiết kế Mức Logic

1.3.1 Cơ bản về Mô hình Quan hệ

- **Cấu trúc:** Lược đồ quan hệ $R(A_1, \ldots, A_n)$; các bộ (tuple) không có thứ tự và không cho phép trùng lặp trong mô hình hình thức.
- **Ràng buộc Toàn vẹn:** Ràng buộc miền giá trị (nguyên tố, có kiểu), ràng buộc khóa (siêu khóa/khóa ứng viên/khóa chính), toàn vẹn thực thể (khóa chính không NULL), toàn vẹn tham chiếu (giá trị khóa ngoại phải xuất hiện trong khóa chính được tham chiếu).

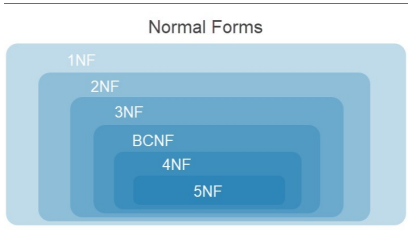
ER-to-Relational Mapping Rules

- **Strong Entity:** Tạo bảng. PK = Entity PK.
- **Weak Entity:** Tạo bảng. PK = (Owner PK + Partial Key). FK to Owner.
- **1:N Relationship:** Đưa PK của bên "1"làm FK vào bảng bên "N".
- **M:N Relationship:** Tạo **bảng trung gian**. PK = (PK_A + PK_B). Cả hai đều là FK.
- **Multivalued Attribute:** Tạo **bảng mới**. PK = (Entity PK + Attribute Value).
- **N-ary (Ternary) Relationship:** Tạo **bảng mới**. PK = (PK_A + PK_B + PK_C). Tất cả đều là FK.

1.3.2 Lý thuyết Chuẩn hóa

- **Mục tiêu:** Giảm thiểu dư thừa, tránh dị thường (Thêm, Xóa, Sửa).
- **Phụ thuộc Hàm (FD):** $X \to Y$ (Nếu $t_1[X] = t_2[X]$ thì $t_1[Y] = t_2[Y]$).
- **Các loại Key:**
  - *Superkey:* Xác định duy nhất một bộ.
  - *Candidate Key:* Superkey tối thiểu.
  - *Prime Attribute:* Thuộc tính nằm trong bất kỳ Candidate Key nào.

Các Dạng Chuẩn (Normal Forms)



(Hierarchy: BCNF ⊂ 3NF ⊂ 2NF ⊂ 1NF)

1NF (Atomic): Miền giá trị nguyên tố.
↪ *Vi phạm:* Thuộc tính đa trị, lồng nhau, lặp lại nhóm.

2NF (No Partial): Là 1NF + Thuộc tính *non-prime* phụ thuộc đầy đủ vào khóa.
↪ *Vi phạm:* $\exists X \subsetneq Key$ sao cho $X \to NonPrime$.
(Chỉ xảy ra nếu Key là khóa phức hợp).

3NF (No Transitive): Là 2NF + Không có phụ thuộc bắc cầu giữa các *non-prime*.
→ *Định nghĩa:* Với mọi $X \to A$ (không tầm thường), phải thỏa: (a) X là Superkey **HOẶC** (b) A là Prime Attribute.

BCNF (Strict): Nghiêm ngặt hơn 3NF.
→ *Định nghĩa:* Với mọi $X \to A$, X **bắt buộc là Superkey**.
(Khác biệt: BCNF không chấp nhận ngoại lệ "A là Prime"như 3NF).

Tính chất Phân rã (Decomposition Properties)

1. **Kết nối bảo toàn thông tin (Lossless Join):** *(Bắt buộc)*
   Để phân rã R thành R_1, R_2 không bị mất dữ liệu, điều kiện là:
   $(R_1 \cap R_2) \to R_1$ **HOẶC** $(R_1 \cap R_2) \to R_2$.
   (Giao của 2 bảng phải là khóa của ít nhất 1 bảng).
2. **Bảo toàn phụ thuộc (Dependency Preservation):**
   Các FD ban đầu có thể được kiểm tra riêng lẻ trên từng R_i mà không cần join lại. (BCNF có thể không bảo toàn phụ thuộc).

1.3.3 Phi chuẩn hóa (Denormalization)

- **Mục tiêu:** Cải thiện hiệu suất đọc bằng cách đưa dư thừa vào lược đồ, ngược với chuẩn hóa.
- **Động lực:** Tránh Join tốn kém; giảm độ phức tạp truy vấn; tăng locality dữ liệu.

Kỹ thuật Denormalization:

- **Materialized Views:** Kết quả truy vấn được tính trước và lưu trữ; cập nhật khi dữ liệu thay đổi.
- **Precomputed Aggregates:** Lưu giá trị tổng hợp (COUNT, SUM) trong bản ghi để tránh tính lại.
  VD: Lưu số email chưa đọc trong bảng User thay vì đếm mỗi lần.
- **Document Databases:** Nhúng (embedding) dữ liệu liên quan trong một document thay vì tham chiếu.
  VD: MongoDB nhúng thông tin Worker vào document Project.
- **Star Schema (DW):** Dimension tables được denormalize (VD: Brand, Category trong dim_product).
- **Microservices:** Sao chép (replicate) dữ liệu giữa các service để tách biệt và giảm phụ thuộc.

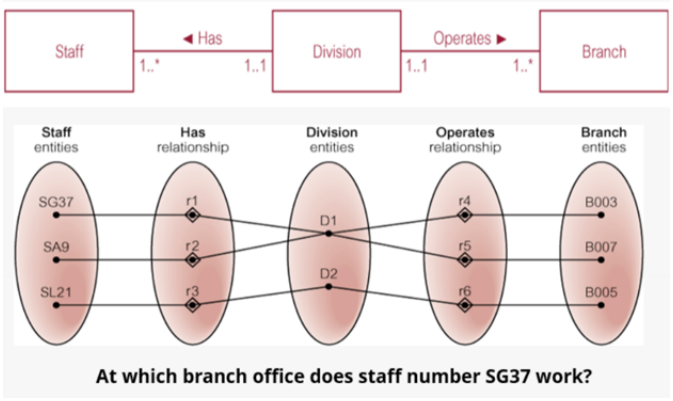
Đánh đổi (Trade-offs):

- **Write Overhead:** Mỗi cập nhật phải sửa tất cả bản sao dư thừa ⇒ ghi chậm hơn, phức tạp hơn.
- **Data Inconsistency:** Rủi ro không nhất quán nếu một bản sao được cập nhật mà bản khác thì không.
- **Storage Cost:** Lưu trữ dữ liệu trùng lặp tốn bộ nhớ hơn.

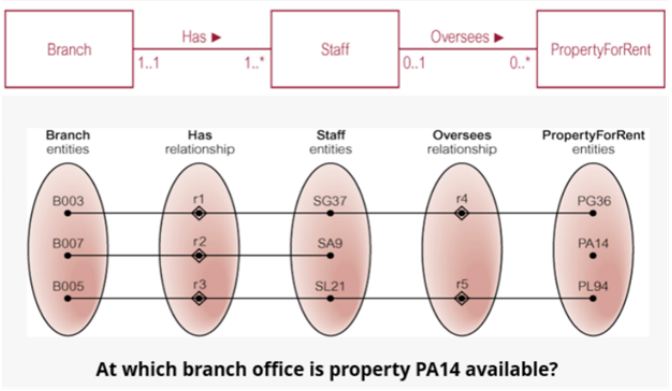
Khi nào dùng: OLAP/DW (đọc nhiều), NoSQL (thiếu join), microservices (tách biệt). *Tránh:* OLTP cần ACID chặt.

1.3.4 Bẫy Thiết kế CSDL (Design Traps)

- **Bẫy Kết nối trong ER:**



(Fan Trap)



(Chasm Trap)

  - *Fan Trap:* Đường dẫn giữa các thực thể mơ hồ do nhiều quan hệ 1:N phân nhánh từ một thực thể. VD: NhanVien → PhongBan → ChiNhanh (không xác định được nhân viên làm ở chi nhánh nào). *Giải pháp:* Thêm quan hệ trực tiếp NhanVien-ChiNhanh.
  - *Chasm Trap:* Đường dẫn không tồn tại do tham gia tùy chọn. VD: KhanhHang → TaiSan → ChiNhanh (nếu tài sản chưa niêm yết thì không liên kết được khách hàng với chi nhánh). *Giải pháp:* Đổi sang tham gia bắt buộc hoặc thêm quan hệ trực tiếp.
  - **Dị thường Cập nhật (Update Anomalies):** Do thiết kế không chuẩn hóa:
    - *Insertion:* Không thể thêm phòng ban mới nếu chưa có nhân viên.
    - *Deletion:* Xóa nhân viên cuối cùng làm mất thông tin phòng ban.
    - *Modification:* Thay đổi tên phòng ban phải cập nhật nhiều bộ.
  - **Bộ giả (Spurious Tuples):** Kết nối các quan hệ phân rã sai (không qua PK/FK hợp lệ) tạo ra bản ghi ảo. *Giải pháp:* Dùng phân rã bảo toàn thông tin (lossless join).
  - **Bẫy NULL:** Quá nhiều thuộc tính NULL lãng phí bộ nhớ và gây khó khăn trong truy vấn tổng hợp.
  - **Sai lầm phổ biến:** Dùng PK của thực thể này làm thuộc tính của thực thể khác thay vì mô hình hóa quan hệ; gán PK vào thuộc tính của quan hệ; dùng thuộc tính đơn trị khi cần đa trị.
  - **Entity Trap (Kiến trúc):** Thiết kế component 1-1 với bảng DB (VD: CustomerManager cho bảng Customer) thay vì theo workflow nghiệp vụ → vi phạm tách biệt dữ liệu trong microservices.

Thực hành tốt: Dùng BCNF/3NF; phân rã bảo toàn thông tin; đảm bảo đường dẫn ER rõ ràng; thiết kế theo hành vi nghiệp vụ, không theo thực thể.

1.4 Nguyên lý Thiết kế Mức Vật lý

- **Kiến trúc Lưu trữ:** Dữ liệu bền vững trên đĩa/SSD trong các **khối (block)** kích thước cố định.
- **Phân tích Tải (Job Mix):** Xác định các quan hệ/tập tin thường truy cập, điều kiện chọn (bằng/khác/khoảng), và tần suất cập nhật so với truy vấn.
- **Cấu trúc Chỉ mục:** Chỉ mục có thứ tự (B+-Trees) và chỉ mục băm; chỉ mục **chính/phân cụm** (quy định thứ tự vật lý; tối đa một trên mỗi tập tin) so với chỉ mục **phụ**.
- **Tối ưu hóa Truy vấn:** Dựa trên chi phí (thống kê) và các quy tắc kinh nghiệm (đẩy phép chọn/chiếu xuống sớm) để chọn kế hoạch hiệu quả.

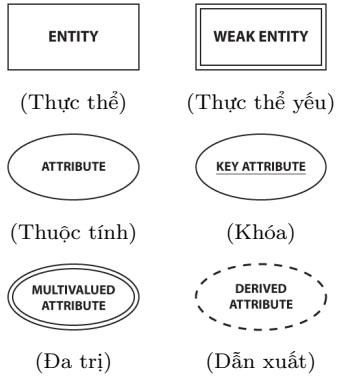
1.5 Các bước vẽ ERD

1. **Xác định Thực thể (Entities):**
   Tìm các *danh từ* (Nouns) quan trọng trong yêu cầu (Vd: Employee, Student). Tránh nhầm lẫn thuộc tính là thực thể.
2. **Xác định Mối kết hợp (Relationships):**
   Tìm các *động từ* (Verbs) kết nối các thực thể (Vd: Works_for, Teaches).
3. **Xác định Thuộc tính (Attributes):**
   Xác định thông tin chi tiết cho mỗi thực thể. Xác định thuộc tính đa trị, dẫn xuất, phức hợp.
4. **Xác định Khóa chính (Primary Keys):**
   Chọn thuộc tính định danh duy nhất cho mỗi thực thể và gạch chân nó.
5. **Xác định Bản số (Cardinality Ratio):**
   Phân tích số lượng tham gia: 1:1, 1:N, hay M:N.
6. **Xác định Ràng buộc tham gia (Participation):**
   Có bắt buộc không? (Total - Nét đôi) hay Tùy chọn? (Partial - Nét đơn).
   Hỏi: 'Thực thể A có thể tồn tại mà không cần B không?'
7. **Vẽ phác thảo & Tinh chỉnh:**
   Vẽ sơ đồ, loại bỏ các thuộc tính dư thừa. Chuyển quan hệ M:N thành thực thể liên kết nếu cần thiết.

1.6 Chen Notation: ER & EER

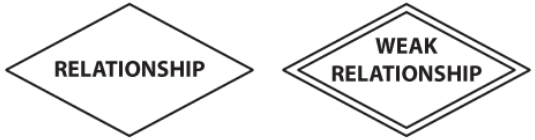
1.6.1 Entities & Attributes

Khi nào dùng: **Thực thể** cho đối tượng độc lập (NhanVien, SanPham, KhanhHang); **Thực thể yếu** cho đối tượng phụ thuộc (NguoiPhuThuoccủa nhân viên, ChiTietDonHang); **Khóa** là định danh duy nhất (MSNV, CCCD); **Đa trị** cho thuộc tính nhiều giá trị (số điện thoại, email); **Dẫn xuất** cho giá trị tính toán (tuổi từ ngày sinh, tổng tiền).



1.6.2 Relationships

Khi nào dùng: **Quan hệ thường** cho liên kết độc lập (NhanVien *làm việc cho* PhongBan, KhanhHang *mua* SanPham); **Quan hệ xác định** khi thực thể yếu phụ thuộc vào thực thể chủ (NguoiPhuThuoc *thuộc về* NhanVien với khóa bộ phận là tên người phụ thuộc).



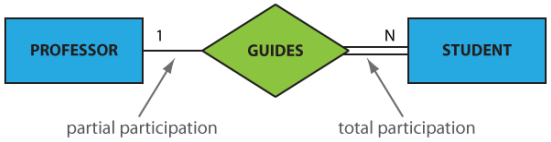
Bao gồm: Quan hệ (Hình thoi), Quan hệ xác định (Thoi đôi)

1.6.3 Ràng buộc (Constraints)

Khi nào dùng: Xác định quy tắc nghiệp vụ giữa các thực thể.

Bản số (Cardinality): 1:1 (NhanVien *quản lý* PhongBan - mỗi phòng có 1 trưởng), 1:N (PhongBan *có* NhanVien - nhiều nhân viên/phòng), M:N (NhanVien *tham gia* DUAN - nhiều-nhiều).

Tham gia (Participation):

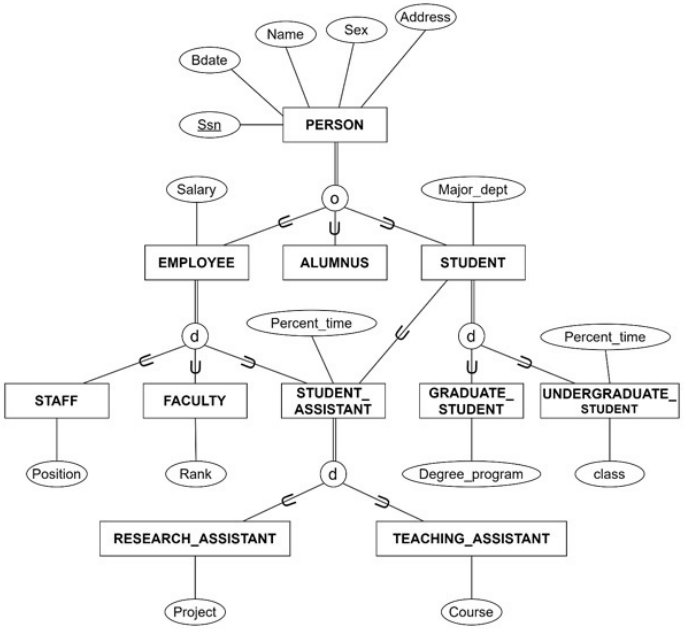


(Partial Participation (Từng phần): Nét đơn
Total Participation (Toàn phần): Nét đôi)

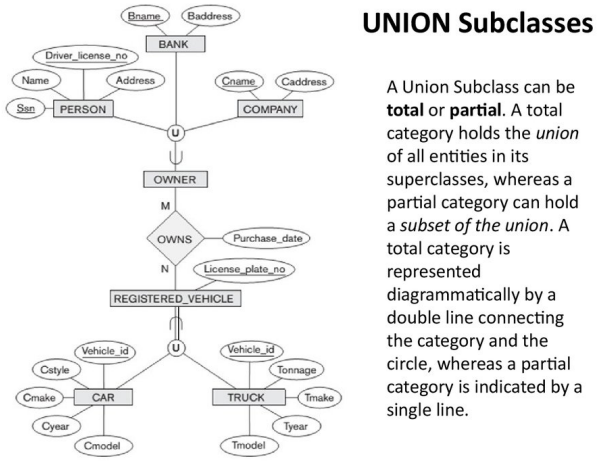
Min-Max (min, max): Ghi cặp số trên cạnh. Ví dụ: NhanVien $(1,1)$ *làm việc cho* $(0,N)$ PhongBan nghĩa là mỗi nhân viên bắt buộc làm việc cho đúng 1 phòng ban, mỗi phòng ban có thể có 0 đến nhiều nhân viên. $(0,1)$: tùy chọn, tối đa 1; $(1,N)$: bắt buộc, có thể nhiều. (An extension of Participation và Cardinality)

1.6.4 EER Chuyên biệt hóa & Tổng quát hóa

Khi nào dùng: **Disjoint** khi lớp con không chồng lấp (NhanVien là KỸ SƯ hoặc QUẢN LÝ, không đồng thời); **Overlapping** khi có thể thuộc nhiều lớp (NGƯỜI là SINH VIÊN và/hoặc NHÂN VIÊN); **Union** khi lớp con kế thừa từ nhiều lớp cha (CHỦ SỞ HỮU có thể là NGƯỜI hoặc CÔNG TY hoặc NGÂN HÀNG). **Total** khi mọi thực thể cha phải thuộc ít nhất 1 lớp con; **Partial** khi không bắt buộc.



(Disjoint & Overlapping)



(Union)

Ký hiệu: Hình tròn (d: disjoint, o: overlapping, U: union), Nét đôi (Total), Nét đơn (Partial).

1.6.5 Ternary Relationships & EER Counting

Mối kết hợp Tam nguyên (Ternary Relationships)

- **Định nghĩa:** Mối kết hợp giữa 3 thực thể (A, B, C) cùng lúc.
- **Quy tắc đọc bản số (Look-Across):** Ký hiệu bản số gần thực thể C cho biết số instances của C liên kết với *một cặp* (A, B).
  VD: Supplier-Project-Device: Bản số gần Device (ví dụ: N) nghĩa là một cặp (Supplier, Project) có thể cung cấp nhiều Device.

EER Counting Rules (Đếm thực thể)

- **Disjoint (d):** Các lớp con không giao nhau.
  $|Superclass| = |Subclass_1| + |Subclass_2| + ... + |Subclass_n|$
- **Overlapping (o):** Các lớp con có thể giao nhau.
  $|Superclass| < |Subclass_1| + |Subclass_2| + ...$ (do có phần giao)
  Áp dụng công thức: $|A \cup B| = |A| + |B| - |A \cap B|$
- **Union (U):** Lớp con là tập con của hợp các lớp cha.
  $|Subclass| \leq |Parent_1 \cup Parent_2 \cup ...|$

2 Xử lý & Tối ưu hóa Truy vấn

Quy trình từ câu lệnh SQL → Kết quả. Trọng tâm: Tính toán chi phí I/O (số block truy cập) để chọn chiến lược tối ưu.

2.1 Ba Giai đoạn Chính (Overview)

1. **Parsing & Translation:** Kiểm tra cú pháp/ngữ nghĩa → Tạo cây đại số quan hệ (Canonical Tree).
2. **Optimization (Quan trọng nhất):** Sử dụng thống kê (Catalog) để ước lượng chi phí → Chọn Execution Plan rẻ nhất (ít I/O nhất).
3. **Evaluation:** Thực thi kế hoạch (dùng Pipelining hoặc Materialization).

2.2 Phase 1: Parsing & Translation

2.2.1 Syntax & Verification

- **Syntax Checking:** Parser validates SQL syntax against language grammar.
- **Semantic Verification:** Verify relation names, attribute names, data types exist in database schema.
- **Authorization:** Check user has necessary privileges (SELECT, INSERT, UPDATE, DELETE) on referenced relations.

2.2.2 Internal Representation

- **Parse Tree:** Construct hierarchical syntax tree representing query structure.
- **Relational Algebra Translation:** Convert SQL to relational-algebra expression.
  Example: `SELECT name FROM instructor WHERE salary > 80000` $\rightarrow \pi_{name}(\sigma_{salary>80000}(instructor))$
- **Query Blocks:** Decompose complex SQL into basic units (`SELECT-FROM-WHERE` expressions) that map to algebraic operators.
  Nested Subqueries: Each subquery becomes separate query block. Correlated subqueries require special handling.
- **View Expansion:** If query references views, replace view names with their defining relational-algebra expressions.
  Recursive Process: Views may reference other views → expand until only base relations remain.

2.2.3 Query Tree (Initial Canonical Form)

- **Structure:** Tree representation where:
  - *Leaf nodes:* Base relations (tables).
  - *Internal nodes:* Relational operations (select σ, project π, join $\bowtie$, etc.).
  - *Execution flow:* Bottom-up evaluation - leaf data flows up through operations to root.
- **Canonical Form:** Initial tree follows SQL structure directly:
  - Cartesian products appear when multiple relations in FROM without explicit join conditions.
  - Selections appear high in tree (near root) reflecting WHERE clause position.
  - Projections at root reflecting SELECT clause.
- **Problem:** Canonical tree often inefficient → requires optimization transformation.

2.3 Phase 2: Tối ưu hóa Truy vấn (Query Optimization)

2.3.1 Tổng quan Quy trình Tối ưu hóa

After parsing, the optimizer transforms the canonical query tree into an efficient execution plan through two complementary approaches:

- **Heuristic Optimization:** Apply algebraic transformation rules to restructure tree (reduce search space).
- **Cost-Based Optimization:** Enumerate candidate plans, estimate costs using statistics, choose optimal plan.

2.3.2 Tối ưu hóa Heuristic (Dựa trên luật)

Mục tiêu: Thu gọn kích thước dữ liệu trung gian càng sớm càng tốt.

Các Quy tắc Cốt lõi (Phải nhớ)

1. **Perform Selections Early (σ down):** Đẩy phép CHỌN xuống thấp nhất có thể.
   Lý do: Giảm số dòng (r) trước khi thực hiện Join (phép toán tốn kém nhất).
2. **Perform Projections Early (π down):** Đẩy phép CHIẾU xuống thấp nhất.
   Lý do: Giảm kích thước mỗi dòng (tuple size), tăng bfr (blocking factor).
3. **Tránh tích Đề-các ($\times$):** Luôn kết hợp $\times$ với σ để thành Join ($\bowtie$).

Transformation Rules

- **Cascading of Selections:** $\sigma_{c1 \wedge c2}(R) = \sigma_{c1}(\sigma_{c2}(R))$.
  Application: Break conjunctive conditions to move parts down different branches.
- **Commutativity:** $\sigma_{c1}(R \bowtie S) = (\sigma_{c1}(R)) \bowtie S$ (if $c1$ only involves R).
  Application: Push selections down past joins.
- **Associativity:** $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.
  Application: Reorder join sequence so most restrictive joins (highest selectivity) occur first.

Heuristic Algorithm (6 steps)

1. **Break up Selections:** Use cascading rule to split conjunctive select conditions.
2. **Move Selections Down:** Push σ down as far as possible (using commutativity).
3. **Reorder Leaf Nodes:** Rearrange leaf nodes (relations) - relation with most restrictive selection (lowest selectivity) first.
4. **Form Joins:** Combine Cartesian products with subsequent selections into Join operations.
5. **Move Projections Down:** Break projection lists, only keep attributes needed for query result or subsequent operations.
6. **Identify Subtrees:** Group operations that can be executed by single algorithm (e.g., single access method).

Trade-offs & Limitations

- **Not Always Optimal:** Heuristics reduce optimization cost but don't guarantee finding optimal plan.
  Example: "Selection early"usually good, but if selection needs to scan large relation while join has efficient index, joining first may be cheaper.
- **Hybrid Approaches:** Commercial optimizers combine both.
  System R: Heuristics (only consider left-deep trees) + cost-based (within those constraints).
  Oracle: Heuristics rank access paths + cost-based chooses join methods.

Subquery vs Join Optimization

- **Subquery Unnesting:** Optimizers generally convert **Nested Subqueries** (especially correlated) into **Joins**.
- **Preference Order:** `JOIN` > `Correlated Subquery` > `Cartesian Product` (worst).
- **Why?** Joins allow optimizer to choose efficient algorithms (Hash/Merge Join), whereas correlated subqueries force "Nested Loop"behavior (inner query runs once per outer row).

2.3.3 Cost-Based Optimization: Foundations

After heuristic optimization, system generates multiple candidate execution plans and estimates cost for each using statistical information from database catalog.

Cost Metrics

- **Primary Metric - Disk I/O:** Number of block transfers + number of disk seeks.
  Reason: For large disk-resident databases, I/O cost dominates CPU cost.
- **Secondary Metrics:** CPU time (for in-memory operations), network communication cost (distributed systems).
- **Response Time vs Throughput:** Systems may optimize for fast first-result (pipelining) or total execution time (materialization).

Cost Function Notation *Ước tính tài nguyên (disk I/O) để xử lý truy vấn: số block transfers (b) và số disk seeks (S).*

- b (hoặc b_r): Tổng số blocks trong file.
- r (hoặc n_r): Tổng số records (tuples) trong relation.
- s (hoặc s_A): **Selection cardinality** - số records trung bình thỏa điều kiện.
- bfr: Blocking factor (số records trên mỗi block).
  $bfr = \lfloor BlockSize(bytes)/RecordSize(bytes) \rfloor$.
- x (hoặc h_i): Số levels trong index (chiều cao B+-tree).
- t_S: Average time cho disk seek.
- t_T: Average time để transfer một block.
- **Công thức tổng:** Cost = $b \times t_T + S \times t_S$ (transfer time + seek time).

Statistical Information (Catalog Metadata)

- **Relation Statistics:** n_r (number of tuples), b_r (number of blocks), l_r (tuple size in bytes).
- **Attribute Statistics:** $V(A, r)$ (number of distinct values for attribute A), $min(A, r)$ and $max(A, r)$ (value range).
- **Index Statistics:** Index type, height (h_i), number of leaf blocks, clustering vs non-clustering.
- **Histograms:** Store distribution of attribute values to improve selectivity estimates.
  Equi-width: Divide value range into equal-sized buckets, count tuples in each.
  Equi-depth: Buckets contain approximately equal number of tuples.
  Use: Handle skewed data distributions (e.g., most employees earn 30K-50K, few earn >100K).

2.4 Cơ bản về Chỉ mục (Index Structures)

Indexes are critical data structures that enable efficient access paths for query execution. Understanding index characteristics is essential for cost estimation and access path selection.

2.4.1 Chỉ mục là gì?

- **Mục đích:** Tăng tốc độ truy xuất dữ liệu bằng cách tạo đường dẫn phụ trợ đến các bản ghi.
- **Khóa tìm kiếm:** (Các) thuộc tính dùng để tìm bản ghi; không nhất thiết là khóa chính; có thể là khóa phức hợp (nhiều cột).
- **Đánh đổi:** Đọc nhanh hơn so với ghi chậm hơn (chi phí bảo trì chỉ mục khi INSERT/UPDATE/DELETE).

2.4.2 Phân loại & Chiến lược Lưu trữ

- **Dense Index (Chỉ mục Đặc):**
  - Có một index entry cho **mỗi record** trong data file.
  - **Sử dụng khi:** File dữ liệu **không** được sắp xếp theo trường đánh index (Secondary Index).
  - **Size:** Số Index Entries $n_i = r$ (số dòng dữ liệu).
- **Sparse Index (Chỉ mục Thưa):**
  - Chỉ có index entry đại diện cho **mỗi block** (hoặc mỗi nhóm).
  - **Sử dụng khi:** File dữ liệu **đã được sắp xếp** theo trường đánh index (Primary/Clustering Index).
  - **Size:** Số Index Entries $n_i = b$ (số block dữ liệu).
  - *Ví dụ (Câu 13 - F1.1):* 70.000 records, bfr=17, sắp xếp theo SSN. Index trên SSN là Sparse.
    $\Rightarrow$ Số entries = $70.000/17 \approx 4.118$.

2.4.3 Các loại Chỉ mục Cơ bản

1. **Primary Index:** Trên trường **Khóa (Unique)** + File **đã sắp xếp**. (Thường là Sparse).
2. **Clustering Index:** Trên trường **Không Khóa (Non-Unique)** + File **đã sắp xếp**. (Sparse - 1 entry cho mỗi giá trị riêng biệt).
3. **Secondary Index:** Trên bất kỳ trường nào + File **không sắp xếp** (hoặc sắp xếp theo trường khác). (Luôn là Dense).

2.4.4 Công thức Số lượng Index Entries

Xác định số hàng trong file chỉ mục (không phải chi phí truy cập).

- **Primary Index (Sparse):** Số Index Entries = Số **Blocks** trong Data File.
  Entries = $b = \lceil r/bfr \rceil$
  Lý do: Mỗi block có 1 entry (anchor record/block anchor).
- **Clustering Index (Sparse):** Số Index Entries = Số **Distinct Values** của clustering field.
  Entries = NDV (Number of Distinct Values)
  Lý do: Mỗi giá trị duy nhất có 1 entry trỏ đến block đầu tiên chứa giá trị đó.
- **Secondary Index (Dense):** Số Index Entries = Số **Records** trong Data File.
  Entries = r
  Lý do: Mỗi record có 1 entry (dense index).

2.4.5 B-Trees & B+-Trees

So sánh

Câu hỏi thường gặp: Tại sao B+-Tree là chuẩn công nghiệp thay vì B-Tree?

B-Tree	**B+-Tree** (Chuẩn DBMS)
Data Pointers: Lưu ở **tất cả** các node (Internal + Leaf).	**Data Pointers:** Chỉ lưu ở **Leaf Node**. Internal node chỉ chứa key để điều hướng.
Search: Có thể dừng ở Internal node nếu tìm thấy key.	**Search:** Luôn phải đi xuống tận Leaf node để lấy dữ liệu.
Fan-out: Thấp hơn (do tốn không gian lưu Data ptr). Cây cao hơn → Nhiều I/O hơn.	**Fan-out: Cao hơn** (Internal node nhẹ). Cây thấp hơn (h nhỏ) → Ít I/O hơn.
Range Query: Khó khăn (phải duyệt cây kiểu trung thứ tự - In-order traversal).	**Range Query: Rất nhanh** (các Leaf node liên kết với nhau thành Linked List).

Các tính chất cốt lõi

- **Balanced Tree:** Mọi đường đi từ gốc đến lá đều có độ dài bằng nhau (h).
- **Root Constraint:** Node gốc luôn chiếm **đúng 1** disk block.
- **Node Capacity:** Mỗi node là 1 disk block (thường là 4KB).
- **Fill Factor:** Các node (trừ root) phải đầy ít nhất 50% ($\lceil p/2 \rceil$). Nếu ít hơn → Underflow (Gộp node).

Công thức tính Fan-out (Order p) [Cần nhớ để tính]

Ký hiệu: Block size B, Tree Pointer P, Key size V, Data Pointer P_r.

- **B+-Tree Internal Node (Tìm p):**
  Node chứa p con trỏ cây và $(p-1)$ khóa. (Không chứa data pointer).
  Công thức: $p \times P + (p-1) \times V \leq B$
- **B+-Tree Leaf Node (Tìm p_{leaf}):**
  Node chứa p_{leaf} cặp (Data Pointer + Key) và 1 Next Pointer.
  Công thức: $p_{leaf} \times (P_r + V) + P \leq B$
- *Lưu ý cho B-Tree thường:* Internal node B-Tree có chứa Data Pointer, công thức sẽ là: $p \times P + (p-1) \times (V + P_r) \leq B$.

Ví dụ Tính toán Sức chứa (Capacity Example)

Đề bài: Block size $B = 512$ bytes; Key size $V = 9$; Data pointer $P_r = 7$; Tree pointer $P = 6$.

Bước 1: Tính Fan-out (Order p)

- **Nút trong B-Tree (B-Tree Internal):** Chứa cả Data Pointer.
  $(p \times 6) + ((p-1) \times (7+9)) \leq 512 \Rightarrow 22p \leq 528 \Rightarrow \mathbf{p = 23}$.
- **Nút trong B+-Tree (B+-Tree Internal):** Chỉ chứa Key.
  $(p \times 6) + ((p-1) \times 9) \leq 512 \Rightarrow 15p \leq 521 \Rightarrow \mathbf{p = 34}$.
  Kết luận: B+-Tree có fan-out cao hơn 48% → cây ngắn hơn.
- **Nút lá B+-Tree (Leaf):**
  $(p_{leaf} \times (7+9)) + 6 \leq 512 \Rightarrow 16 \times p_{leaf} \leq 506 \Rightarrow \mathbf{p_{leaf} = 31}$.

Bước 2: Ước lượng Tổng sức chứa (Giả sử cây cao 3 mức, đầy 69%)

- **B-Tree:** Tổng entry $\approx \mathbf{4.095}$. (Do p nhỏ).
- **B+-Tree:** Tổng entry $\approx \mathbf{255.507}$. (Do p lớn, số lượng lá tăng theo lũy thừa).
  → **Hiệu quả:** B+-Tree lưu được gấp 60 lần dữ liệu với cùng chiều cao cây.

Thao tác Cập nhật (Update Operations)

Cơ chế duy trì tính cân bằng của cây.

- **Splitting (Khi Insert):** Nếu node bị đầy ($> B$), tách thành 2 node. Key ở giữa (middle key) được đẩy lên (promote) node cha.
  → Có thể làm tăng chiều cao cây ($h+1$) nếu lan truyền đến Root.
- **Merging (Khi Delete):** Nếu node bị rỗng ($< 50\%$), mượn key từ anh em hoặc gộp 2 node lại.
  → Có thể làm giảm chiều cao cây ($h-1$).

Implementation & Optimizations

- **Write-Ahead Log (WAL):** B-Trees overwrite pages on disk. Crash during multi-page update (split) → corruption.
  → WAL writes modifications before updating tree pages → durability & atomicity.
- **Concurrency (Latches):** Multiple threads access index concurrently → use latches (lightweight locks).
  Latch Crabbing: Lock parent → lock child → release parent → high concurrency.
- **Bulk Loading:** Inserting records one-by-one → random I/O → inefficient.
  → Bulk load: Sort data → build tree bottom-up (sequential writes) → create parent nodes as needed.
- **Flash Storage (SSD):** Random reads fast, random writes expensive (erase-modify-write cycles).
  → LSM-Tree (Log-Structured Merge-Tree) preferred for write-heavy workloads, but B-Trees remain general-purpose standard.
- **Prefix Compression:** Internal nodes only store prefix of key (enough to distinguish subtrees) instead of full key → increases fan-out.
- **Right-Only Appends:** With auto-incrementing keys, allocate new node when rightmost leaf full instead of split → avoids half-empty pages.

2.4.6 Advanced Index Types

- **Composite Indexes:** Multi-column keys (e.g., (City, LastName)); support leftmost prefix queries (City), (City, LastName); order columns by selectivity.
- **Function-Based Indexes:** Indexes on expressions (e.g., `LOWER(email)`); queries must use the exact function to utilize the index.
  - *Problem:* Standard index on `Name` is **useless** for `WHERE lower(Name) = '...'`.
  - *Solution:* `CREATE INDEX idx_lower ON Employee (lower(Name));`
  - *Key Concept:* The index stores the *result* of the function, allowing the DB to search directly on the computed values.

2.4.7 Chỉ mục Băm & Bitmap

Chỉ mục Băm (Hash Indexes)

- **Sử dụng:** Tìm kiếm chính xác cực nhanh (truy vấn điểm); không hỗ trợ khoảng.
- **Xử lý đụng độ:** Dùng danh sách liên kết (chaining) với bucket tràn.
- **Biến thể động:** Băm mở rộng/tuyến tính (Extendible/Linear hashing) tăng trưởng dần mà không cần xây lại toàn bộ.

Chỉ mục Bitmap *Tối ưu cho thuộc tính có độ chọn lọc thấp (ít giá trị riêng biệt).*

- **Cấu trúc:** Đánh số thứ tự bản ghi (0, 1, 2, ...); mỗi giá trị riêng biệt có một bitmap; bit $i = 1$ nếu bản ghi i có giá trị đó.
- **Ví dụ (bảng 5 dòng):**
  - `gender='m'`: 10010 `gender='f'`: 01101
  - `income='L1'`: 11000 `income='L2'`: 00100
- **Truy vấn: gender='f' AND income='L2'**
  01101 AND 00100 = 00100 ⇒ bản ghi 2
- **Ưu điểm:** Gọn nhẹ (1 triệu dòng = 125 KB mỗi bitmap); thao tác bitwise nhanh; hiệu quả cho bộ lọc nhiều điều kiện; hỗ trợ COUNT qua đếm bit.

Chiến lược chọn Index (Heuristics)

- **Truy vấn khoảng (>, <, BETWEEN):** Bắt buộc dùng **B+ Tree**. (Hash index vô dụng).
- **Truy vấn chính xác (=): Hash Index** tốt nhất ($O(1)$), sau đó đến B+ Tree.
- **Update nhiều:** Hạn chế tạo quá nhiều Index (vì tốn chi phí cập nhật cây).
- **Thuộc tính độ chọn lọc thấp (Giới tính, True/False):** Dùng **Bitmap Index**.

2.5 Hàm Chi phí & Kỹ thuật Chỉ mục

2.5.1 Index Properties & Calculations

- **Dense Index:** Mỗi record có 1 index entry (dùng cho Secondary Index hoặc Primary Index trên file chưa sắp xếp).
- **Sparse Index:** Mỗi block dữ liệu có 1 index entry (chỉ dùng cho Primary/Clustering Index trên file **đã sắp xếp**).
- **Số lượng Index Entries (n_i):**
  - Sparse: $n_i = b$ (số data blocks).
  - Dense: $n_i = r$ (số records).
- **Số block của file Index (b_i):** $b_i = \lceil n_i / bfr_i \rceil$.
- **Index Levels (x):** $x = \lceil \log_{fan\_out}(b_i) \rceil$.

2.5.2 Selection Cost Functions (σ)

Ký hiệu: b: data blocks, r: records, x: index levels, s: selection cardinality (số dòng thỏa điều kiện).

S1: Linear Search (File Scan)

Quét toàn bộ file (Heap file hoặc file không có index phù hợp).
- **Tìm Unique (Key):** $b/2$ (Trung bình).
- **Tìm Non-Unique/Range:** b (Phải quét hết).
- **Khi nào dùng?** File nhỏ hoặc Selectivity thấp (s lớn, >15-20% số dòng).

S2: Binary Search (Sorted File)

Chỉ áp dụng nếu file đã sắp xếp theo trường tìm kiếm (A).
$\mathbf{C_{S2} = \lceil \log_2 b \rceil + \lceil s/bfr \rceil - 1}$
Lưu ý: Ít dùng trong DB thực tế, thường dùng Primary Index thay thế.

S3a: Primary Index (Unique Key) [Quan trọng]

File sắp xếp theo Key + B+-tree.
$\mathbf{C_{S3a} = x + 1}$
Exam Tip: Nếu đề cho chiều cao cây (height) = h, thì $x = h$. Chi phí = Traversing Tree + 1 I/O lấy data.

S3b: Hash Index (Equality Only)

Chỉ dùng cho so sánh bằng (=).
$\mathbf{C_{S3b} \approx 1}$.
Cảnh báo Exam (Q9 - F1.1): Hash index **KHÔNG** hỗ trợ Range Query ($>, <, \geq$). Nếu query là 'WHERE Dno > 10', Hash Index vô dụng → Dùng B+-tree hoặc Scan.

S5: Clustering Index (Non-Unique, Sorted File)

File sắp xếp theo trường tìm kiếm (không phải Key). Records nằm liền nhau.
$\mathbf{C_{S5} = x + \lceil s/bfr \rceil}$
Hiệu quả: Tốt nhất cho Range Query hoặc Non-Unique Equality.

S6a: Secondary Index (Equality) [Hay gặp nhất]

File **không** sắp xếp theo trường này.
$\mathbf{C_{S6a} = x + s}$ (hoặc $x + s + 1$ nếu tính cả block đầu).
Worst case: Mỗi record nằm ở 1 block khác nhau → s lần Random I/O.
So sánh (Q16 - F1.1): Nếu truy vấn trả về 1 dòng ($s = 1$) → Cost = $x + 1$. Rất rẻ so với Linear Scan (b).

S6b: Secondary Index (Range Query)

Dùng B+-tree trên trường phụ.
$\mathbf{C_{S6b} = x + (\text{leaf blocks}/2) + s}$
Exam Trap: Nếu Range rộng (lấy > 30% dữ liệu), Optimizer sẽ bỏ qua Index và chọn Linear Scan (S1) vì Random I/O quá đắt.

2.5.3 Bảng Tra Cứu Nhanh

Algorithm	Mô tả	Block Accesses
S1	Linear Search	b
S2	Binary Search	$\lceil \log_2 b \rceil + \lceil s/bfr \rceil - 1$
S3a	Primary Index (Unique)	$x + 1$
S5	Clustering Index (Equality)	$x + \lceil s/bfr \rceil$
S6a	Secondary Index (Equality)	$x + s + 1$
S6b	Secondary Index (Range)	$x + (b_{I1}/2) + (r/2)$

2.5.4 Hàm Chi phí cho Phép Kết (JOIN Operations)

Ký hiệu (Notation)

- b_r, b_s: Số block của quan hệ R và S.
- n_r, n_s: Số dòng (tuples) của quan hệ R và S.
- n_B: Số block bộ đệm (memory buffer) khả dụng.
- **Nguyên tắc vàng:** Luôn chọn quan hệ nhỏ hơn (ít block hơn) làm vòng lặp ngoài (Outer Loop) để tối ưu hóa bộ đệm.

J1: Block Nested-Loop Join (Quét toàn bộ)

Dùng khi không có Index nào phù hợp. Đọc từng block của R (ngoài), quét toàn bộ S (trong).
- **Trường hợp Tệ nhất (Worst Case - 3 buffers):**
  $\mathbf{Cost = b_r + (b_r \times b_s)}$ block transfers.
  Giải thích: Đọc mỗi block R 1 lần, với mỗi block R phải đọc lại toàn bộ S.
- **Trường hợp Có Buffer ($n_B > 3$):**
  $\mathbf{Cost = b_r + \lceil b_r/(n_B - 2) \rceil \times b_s}$
  Exam Tip (Đề Tham Khảo Câu 2.2): Hệ số $\lceil b_r/(n_B - 2) \rceil$ là số lần phải quét lại bảng S. Nếu n_B càng lớn, chi phí càng giảm về gần $b_r + b_s$.

J2: Indexed Nested-Loop Join (Index trên bảng TRONG)

Bảng ngoài (R) quét tuần tự, bảng trong (S) dùng Index để tìm khớp.
Điều kiện: Bảng S phải có Index trên thuộc tính Join.
- **Công thức Tổng quát:**
  $\mathbf{Cost = b_r + n_r \times}$ (Cost per Probe)
  Lưu ý: Phụ thuộc vào số dòng (n_r) của bảng ngoài, không phải số block (b_r). Do đó, bảng ngoài nên có ít dòng.

- **Chi tiết Cost per Probe:**
  - *Primary Index (S):* $x + 1$ (Rất rẻ).
  - *Secondary Index (S):* $x + s$ (Đắt hơn nếu s lớn).
  - *Hash Index (S):* ≈ 1 (Rất rẻ cho Equi-Join).
- **So sánh với J1 (Câu 11 - F1.1):** Optimizer thường chuyển Cross-Join (J1) thành Theta-Join/Equi-Join (J2) để tận dụng Index, giảm chi phí từ $b_r \times b_s$ xuống tuyến tính theo n_r.

J3: Sort-Merge Join (Dữ liệu đã sắp xếp)

Hiệu quả nhất nếu cả hai bảng đã được sắp xếp theo thuộc tính Join.

- **Chi phí (Đã sắp xếp): $\mathbf{b_r + b_s}$**. (Đọc mỗi bảng 1 lần).
- **Chi phí (Chưa sắp xếp):** Cost(Sort R) + Cost(Sort S) + $b_r + b_s$.
- **Khi nào dùng?** Khi kết quả cần sắp xếp (ORDER BY) hoặc dữ liệu đã có sẵn thứ tự (Clustered Index).

J4: Hash Join (In-Memory Partitioning)

Chỉ dùng cho Equi-Join (=). Partition bảng R và S vào các bucket băm.

- **Chi phí (Standard): $\mathbf{3(b_r + b_s)}$**.
  Giải thích: 1. Đọc R, S để partition (ghi ra đĩa) → 2. Đọc lại partition để join. Tổng cộng đọc + ghi + đọc = 3 lần.
- **Điều kiện:** Partition của bảng nhỏ phải vừa đủ bộ nhớ (Memory).

2.5.5 Ví dụ Tính Toán: EMPLOYEE ⋈ DEPARTMENT

Thông số:

- **EMP (E):** $n_E = 10{,}000$, $b_E = 2{,}000$. (Bảng lớn)
- **DEPT (D):** $n_D = 125$, $b_D = 13$. (Bảng nhỏ)
- **Index:** Primary Key trên *D.Dnumber* ($x = 1$). Foreign Key trên *E.Dno* (Secondary, $x = 2$).

Kịch bản 1: Block Nested Loop (Không dùng Index)

- *D là Outer (Tối ưu):* $b_D + (b_D \times b_E) = 13 + (13 \times 2{,}000) = \mathbf{26{,}013}$.
- *E là Outer (Tệ):* $b_E + (b_E \times b_D) = 2{,}000 + (2{,}000 \times 13) = 28{,}000$.
  → Chọn bảng nhỏ làm Outer tiết kiệm được một chút, nhưng vẫn rất đắt.

Kịch bản 2: Indexed Nested Loop (Tận dụng Index)

- *Dùng Primary Index trên D (E là Outer):*
  Quét E (b_E) + Tra cứu Index D cho mỗi dòng E ($n_E \times (x + 1)$).
  $2{,}000 + 10{,}000 \times (1 + 1) = \mathbf{22{,}000}$ blocks. (Vẫn đắt do n_E quá lớn).
- *Dùng Secondary Index trên E (D là Outer - **BEST J2**):*
  Quét D (b_D) + Tra cứu Index E cho mỗi dòng D ($n_D \times \text{Cost}(E)$).
  Cost tìm E cho 1 *Dno*: Index ($x = 2$) + Data ($s = 80$ block ngẫu nhiên) = 82.
  Tổng: $13 + 125 \times 82 = 13 + 10{,}250 = \mathbf{10{,}263}$ blocks.
  → **Kết luận:** Dùng bảng nhỏ (D) làm Outer + Index trên bảng lớn (E) hiệu quả gấp đôi cách ngược lại.

Kịch bản 3: Hash Join vs Sort-Merge

- *Sort-Merge (Chưa sort):* Cần sort E ($2000 \log 2000$) rất đắt. Không chọn trừ khi đã sort sẵn.
- *Hash Join:* $3 \times (2{,}000 + 13) = \mathbf{6{,}039}$ blocks.
  → **Hash Join** chiến thắng tuyệt đối nếu bộ nhớ đủ chứa 13 block của D.

2.5.6 Tối ưu hóa Thứ tự Phép Kết (Join Order Optimization)

Mục tiêu: Tìm kế hoạch thực thi có chi phí thấp nhất trong số n! thứ tự kết nối có thể.

Quy hoạch Động (Phương pháp System R)

Thuật toán chuẩn được hầu hết các bộ tối ưu hóa (optimizers) sử dụng.

- **Bước 1:** Tìm kế hoạch tốt nhất để truy cập từng quan hệ đơn lẻ (Quét chỉ mục vs Quét toàn bộ).
- **Bước 2:** Tìm kế hoạch tốt nhất để kết nối các cặp quan hệ ($R \bowtie S$).
- **Bước 3:** Tìm kế hoạch tốt nhất để kết nối 3 quan hệ sử dụng kết quả từ Bước 2.
- **Nguyên lý Tối ưu (Principle of Optimality):** Một kế hoạch tối ưu cho ($R \bowtie S \bowtie T$) bắt buộc phải chứa một kế hoạch tối ưu cho phép kết con ($R \bowtie S$).

Cây Nghiêng Trái (Left-Deep Trees) [Khái niệm Quan trọng]

Các bộ tối ưu hóa thường giới hạn không gian tìm kiếm vào các **Left-Deep Trees**.

- **Cấu trúc:** Nút con bên *phải* của bất kỳ nút kết nối (join node) nào luôn là một **Bảng Cơ sở** (Base Table) (không phải là kết quả trung gian).
- **Tại sao lại là Left-Deep? (Câu hỏi thi):**
  - **Tận dụng Chỉ mục:** Vì quan hệ trong (bên phải) là một bảng cơ sở, ta có thể sử dụng các **Chỉ mục (Indexes)** hiện có cho thuật toán Nested-Loop Joins.
  - **Pipelining (Đường ống):** Cho phép thực hiện pipelining hiệu quả (dữ liệu đi lên từ phía trái, thăm dò/đối chiếu với phía phải).
  - **Không gian Tìm kiếm:** Giảm độ phức tạp từ $O(4^n)$ (Cây rậm - Bushy) xuống $O(n!)$.
- **Đánh đổi:** Có thể bỏ lỡ kế hoạch tối ưu tuyệt đối (ví dụ: nếu một cây Bushy thực sự rẻ hơn), nhưng quá trình tối ưu hóa nhanh hơn nhiều.

2.6 Phase 3: Query Evaluation (Thời gian chạy/Runtime)

2.6.1 Mô hình Iterator (Mô hình Volcano)

Giao diện chuẩn cho tất cả các toán tử (Select, Join, Sort).

- **open():** Khởi tạo trạng thái (ví dụ: cấp phát bộ đệm).
- **next():** Trả về **từng bộ (tuple)** (hoặc một lô) tại một thời điểm.
- **close():** Giải phóng tài nguyên.
- *Lợi ích:* Cho phép Pipelining (các toán tử kéo dữ liệu theo nhu cầu).

2.6.2 Materialization vs. Pipelining [So sánh trong Đề thi]

1. Materialization (Vật chất hóa/Lưu trữ tạm)

Ghi kết quả của mỗi phép toán vào một **Bảng Tạm (Temporary Table)** trên đĩa trước khi bắt đầu phép toán tiếp theo.

- **Ưu điểm:** Đơn giản; Checkpoint/Khởi động lại dễ dàng hơn; Tiết kiệm bộ nhớ.
- **Nhược điểm: Chi phí I/O cao** (Ghi tạm + Đọc tạm); Độ trễ cao (phải đợi kết quả đầy đủ).
- **Khi nào dùng?** Khi kết quả trung gian quá lớn so với bộ nhớ hoặc cần được sắp xếp.

2. Pipelining (Xử lý tức thì/Trên đường truyền)

Truyền các bộ dữ liệu từ nút con sang nút cha ngay lập tức thông qua bộ đệm bộ nhớ.

- **Ưu điểm: Không có Disk I/O** cho kết quả trung gian; Độ trễ thấp (hàng đầu tiên xuất hiện ngay lập tức).
- **Nhược điểm:** Yêu cầu nhiều bộ nhớ hoạt động hơn (các bộ đệm).
- **Pipeline Breakers (Các toán tử gây chặn):** Các phép toán **không thể** pipeline vì chúng cần *toàn bộ* đầu vào trước khi tạo ra hàng kết quả đầu tiên.
  - **Sort (Sắp xếp):** Phải đọc tất cả các hàng để tìm hàng nhỏ nhất.
  - **Aggregate (GROUP BY):** Phải đọc tất cả các hàng trong một nhóm để tính tổng/đếm.
  - **Hash Join (Giai đoạn Build):** Phải xây dựng bảng băm đầy đủ trước khi thăm dò (probing).

2.6.3 Bảng Tra Cứu Nhanh Kế hoạch Đánh giá

Đặc điểm	Materialization	Pipelining
Chi phí I/O	Cao (Ghi bảng tạm)	Thấp (Chỉ dùng bộ nhớ)
Độ trễ (Latency)	Cao (Đợi tất cả)	Thấp (Hàng đầu tiên nhanh)
Bộ nhớ	Thấp (Từng phép toán một)	Cao (Các phép toán đồng thời)
Trường hợp dùng	Sắp xếp phức tạp / Big Data	Truy vấn thời gian thực

3 Big Data & Data Engineering

3.1 Đặc trưng Big Data (The Vs)

- **5V Cốt lõi (Core):**
  - **Volume:** Dung lượng khổng lồ (TB, PB, ZB).
  - **Velocity:** Tốc độ sinh ra & xử lý (Batch → Streaming).
  - **Variety:** Đa dạng định dạng (Structured, JSON, Video, Log).
  - **Veracity:** Độ tin cậy, tính xác thực (Messy/Noisy data).
  - **Value:** Giá trị chuyển hóa thành lợi ích kinh doanh.
- **Các V Mở rộng (Extended):**
  - **Variability:** Tính biến thiên (Ý nghĩa dữ liệu thay đổi theo ngữ cảnh/thời gian).
  - **Validity:** Tính hợp lệ (Dữ liệu có đúng định dạng/chuẩn để dùng không).
  - **Vulnerability:** Tính bảo mật (Dễ bị tấn công/rò rỉ).
  - **Volatility:** Độ bay hơi (Thời gian lưu trữ trước khi xóa/lưu trữ lâu dài).
  - **Visualization:** Khả năng trực quan hóa (Để con người hiểu được).

3.2 Paradigm: Batch vs Streaming

Hai mô hình xử lý dữ liệu cơ bản trong Big Data.

	Batch Processing	Streaming Processing
Đặc trưng	Xử lý dữ liệu tĩnh, lượng lớn theo lô.	Xử lý dữ liệu động, liên tục theo thời gian thực.
Độ trễ	Cao (minutes - hours).	Thấp (seconds - milliseconds).
Công cụ	Hadoop MapReduce, Apache Spark (Batch mode).	Apache Flink, Spark Streaming, Kafka Streams.
Use Case	ETL, báo cáo cuối ngày, ML training.	Real-time analytics, fraud detection, monitoring.
Ưu điểm	Xử lý hiệu quả khối lượng lớn, đơn giản.	Phản hồi nhanh, phát hiện sự kiện ngay lập tức.
Nhược điểm	Không real-time, lãng phí khi data nhỏ.	Phức tạp, khó debug, cần xử lý out-of-order.

Kiến trúc Lai (Hybrid):

- **Lambda Architecture:** Batch layer (chính xác) + Speed layer (real-time) + Serving layer. Phức tạp, duy trì 2 code base.
- **Kappa Architecture:** Chỉ dùng Streaming (đơn giản hóa). Mọi dữ liệu qua stream processor, replay từ Kafka khi cần.

3.3 Partitioning & Replication

3.3.1 Sao chép (Replication)

Mục đích: High Availability (HA) và giảm độ trễ đọc.

- **Single-Leader (Master-Slave):** Mọi ghi vào Leader, Leader chép sang Followers. *Dễ nhất quán, nhưng Leader là nút cổ chai.*
- **Multi-Leader:** Nhiều node chấp nhận ghi. *Tốt cho đa trung tâm dữ liệu, nhưng khó xử lý xung đột.*
- **Leaderless (Dynamo-style):** Ghi/Đọc gửi tới nhiều node. Dùng cơ chế **Quorum** để xác nhận:
  $w + r > n$ (Write nodes + Read nodes > Total replicas) → Đảm bảo đọc thấy dữ liệu mới nhất.

3.3.2 Phân mảnh (Partitioning/Sharding)

Mục đích: Scalability (Mở rộng dung lượng/băng thông).

- **Key Range Partitioning:** Chia theo khoảng khóa (A-C, D-F).
  → *Ưu:* Query theo khoảng (Range scan) hiệu quả.
  → *Nhược:* Dễ bị *Hotspot* (nếu user dồn vào vần A).
- **Hash Partitioning:** Băm khóa để chia đều ($hash(key)\%N$).
  → *Ưu:* Phân phối đều, tránh Hotspot.
  → *Nhược:* Mất khả năng Range Query (phải quét tất cả).

3.4 Định dạng Lưu trữ (File Formats)

Lựa chọn định dạng ảnh hưởng trực tiếp đến hiệu năng đọc/ghi.

3.4.1 Row-based vs. Column-based

- **Row-oriented (CSV, Avro):**
  - Lưu trữ tuần tự từng dòng.
  - *Ưu điểm:* Ghi nhanh (append), tốt khi truy xuất toàn bộ thông tin của 1 entity (OLTP).
  - *Nhược điểm:* Chậm khi tính toán tổng hợp (SUM, AVG) vì phải đọc cả dữ liệu không cần thiết.
- **Column-oriented (Parquet, ORC):**
  - Lưu trữ riêng biệt từng cột.
  - *Ưu điểm:* Nén cực tốt (do dữ liệu cùng kiểu), tối ưu cho OLAP (chỉ đọc cột cần thiết).
  - *Nhược điểm:* Ghi chậm, update tốn kém.

3.4.2 So sánh Avro, Parquet, ORC

Đặc điểm	Avro	Parquet
Mô hình	Row-based	Column-based
Schema	JSON (lưu trong file)	Binary (footer)
Tối ưu cho	Ghi nhiều (Write heavy)	Đọc nhiều (Read heavy)
Schema Evo	Rất tốt (Thêm/bớt field)	Hạn chế
Ecosystem	Kafka, Hadoop	Spark, Impala, Presto

3.5 Hadoop Ecosystem

Open-source framework for distributed storage and processing.

1. **HDFS (Storage):** Distributed file system.
   - *NameNode:* Manages metadata (block locations).
   - *DataNode:* Stores actual data blocks.
   - *Mechanism:* Splits files into blocks (128MB), replicates (x3) for fault tolerance.
2. **YARN (Resource Management):** "Operating system"of the cluster.
   - Distributes resources (RAM, CPU) to applications.
   - Allows multiple engines (Spark, MapReduce) to run on the same cluster.
3. **MapReduce (Processing):** Batch processing model - divide and conquer on a distributed cluster.
   - *Map:* Chia nhỏ & Gán nhãn. Input → Split → `<Key, Value>`.
   - *Shuffle:* Xáo trộn & Gom nhóm. Chuyển dữ liệu qua mạng, gom cùng Key.
   - *Reduce:* Tổng hợp. Xử lý danh sách Value của mỗi Key.

3.6 Công nghệ NoSQL (Storage Tech)

Các mô hình NoSQL cho use case khác nhau.

MongoDB (Document Store)

- **Mô hình:** Schema-on-read, lưu trữ JSON/BSON documents. Collections thay vì tables.
- **Ưu điểm:** Linh hoạt schema (mỗi doc có cấu trúc khác nhau), dễ scale horizontal (sharding), query mạnh (aggregation pipeline).
- **Architecture:** Replica Sets (HA), Sharding (scale-out), WiredTiger storage engine.
- **Use Case:** CMS, Mobile apps, Catalog, Real-time analytics. *VD: Forbes, eBay, Uber.*
- **Trade-offs:** Không ACID cross-document (trước v4.0), chiếm RAM nhiều.

Redis (Key-Value Store)

- **Mô hình:** In-memory key-value, data structures (String, Hash, List, Set, Sorted Set).
- **Ưu điểm:** Cực nhanh (< 1ms latency), atomic operations, support pub/sub, Lua scripting.
- **Persistence:** RDB (snapshot) hoặc AOF (append-only log). Có thể dùng cả 2.
- **Use Case:** Caching (session, query result), Message Queue (Celery), Leaderboard, Rate limiting. *VD: Twitter, GitHub, Stack Overflow.*
- **Trade-offs:** Giới hạn RAM, single-threaded (1 core), không có query phức tạp.

Cassandra (Wide-Column Store)

- **Mô hình:** Wide-column, mỗi row có thể có số cột khác nhau. Organize theo Column Family.
- **Ưu điểm:** Ghi cực nhanh (LSM Tree), linear scalability, masterless (P2P), multi-datacenter replication.

- **Architecture:** Consistent hashing (ring), tunable consistency (quorum), compaction strategies.
- **Use Case:** Time-series data, IoT sensor logs, Event logging, Messaging. *VD: Netflix, Apple, Instagram.*
- **Trade-offs:** Đọc chậm hơn (nhiều SSTable), không join, modeling phức tạp (query-first design).

Neo4j (Graph Database)

- **Mô hình:** Nodes (entities) + Relationships (edges) + Properties. Native graph storage.
- **Ưu điểm:** Traversal cực nhanh (follow pointers), query trực quan (Cypher), ACID transactions.
- **Architecture:** Index-free adjacency (mỗi node chứa pointer đến neighbors).
- **Use Case:** Social networks, Recommendation engines, Fraud detection, Knowledge graphs. *VD: LinkedIn, Walmart, eBay.*
- **Trade-offs:** Scale khó hơn NoSQL khác, không tốt cho bulk data processing.

3.7 Batch Processing

3.7.1 Apache Spark (Unified Analytics Engine)

Thay thế MapReduce với tốc độ cao hơn 100x (in-memory).

- **Core Concept:** RDD (Resilient Distributed Dataset) - immutable, partitioned, parallel.
  - *Transformations:* Lazy (map, filter, join) - tạo DAG.
  - *Actions:* Eager (collect, count, save) - trigger execution.
- **Components:**
  - *Spark SQL:* Query structured data (DataFrame/Dataset API).
  - *Spark Streaming:* Micro-batch streaming (DStream).
  - *MLlib:* Machine learning library (classification, clustering, etc.).
  - *GraphX:* Graph processing (PageRank, connected components).
- **Ưu điểm:** In-memory caching, lazy evaluation, DAG optimization, unified API (batch + streaming).
- **Nhược điểm:** Tốn RAM, không true streaming (micro-batch), overhead cho job nhỏ.
- **Use Case:** ETL, ML training, interactive analytics, log processing. *VD: Netflix, Uber, Airbnb.*

3.7.2 Big Data Warehousing / SQL-on-Hadoop

Các công cụ SQL Analytics cho Big Data - phân biệt với NoSQL operational stores.

- **Apache Hive:** SQL-on-Hadoop, metadata layer trên HDFS. Batch-oriented, high latency.
- **Google BigQuery:** Serverless, columnar storage (Dremel), auto-scaling, pay-per-query.
- **AWS Redshift:** Managed columnar DW, massively parallel processing (MPP).
- **Snowflake:** Cloud-native DW, tách biệt storage/compute, near-zero admin.
- **Presto/Trino:** Distributed SQL query engine, federated queries across data sources.

Phân biệt: Các công cụ trên dùng cho **Analytics/OLAP**, khác với NoSQL (MongoDB, Cassandra) dùng cho **Operational/OLTP**.

3.8 Streaming Processing

Xử lý dữ liệu liên tục, độ trễ thấp (Real-time).

3.8.1 Các chiến lược xử lý (Strategies)

- **Thời gian (Time Domain):**
  - *Event Time:* Thời gian sự kiện xảy ra (quan trọng nhất).
  - *Processing Time:* Thời gian hệ thống nhận được dữ liệu.
  - *Watermark:* Cơ chế xử lý độ trễ (data đến muộn) trong Event Time.
- **Cửa sổ (Windowing):**
  - *Tumbling:* Cố định, không chồng (vd: mỗi 5p).
  - *Hopping/Sliding:* Có chồng lấp (vd: 5p, trượt mỗi 1p).
  - *Session:* Dựa trên hoạt động người dùng (hết timeout thì đóng).
- **Đảm bảo (Guarantees):**
  - *At-most-once:* Gửi 1 lần, chấp nhận mất (vd: Log).
  - *At-least-once:* Không mất, chấp nhận trùng lặp.
  - *Exactly-once:* Chính xác 1 lần (Khó nhất, cần Flink/Kafka).

3.9 Giao thức IoT (IoT Protocols)

3.9.1 MQTT (Message Queue Telemetry Transport)

Design Philosophy

Được thiết kế cho **constrained devices (thiết bị hạn chế)** trong môi trường mạng khó khăn:

- **Low bandwidth (băng thông thấp):** Minimized data packets (gói dữ liệu tối thiểu).
- **High latency (độ trễ cao):** Tối ưu cho mạng chậm.
- **Unreliable networks (mạng không ổn định):** Xử lý mất kết nối, chập chờn.
- **Low power usage:** Tiết kiệm pin cho IoT devices.
- **Real-time processing:** Hỗ trợ xử lý real-time cho streaming data.

Architecture: Publish-Subscribe Model

Khác với client-server truyền thống (client connect trực tiếp client), MQTT dùng **pub-sub model**.

- **Broker:** Central component nhận published messages và deliver đến subscribers phù hợp.
  Flow: Client gửi `CONNECT` → Broker trả `CONNACK`.
- **Publisher:** Client publish message đến broker với specific topic.
- **Subscriber:** Client listen incoming messages trên particular topics.
- **Advantage:** Decoupling (tách rời) - publisher không cần biết subscriber, scale dễ dàng.

Topics (Routing Mechanism)

Structure: Hierarchical strings với topic levels phân cách bởi `/`.

Type	Pattern	Matches & Description
Exact	`area1/humidity`	Chỉ match chính xác topic này. *Use:* Specific sensor.
Single-level	`area1/+`	`+` matches *1* level bất kỳ. ✓ `area1/humidity`, `area1/light` × `area1/sensors/temp`
Multi-level	`area1/#`	`#` matches *tất cả* levels. ✓ `area1/humidity`, `area1/sensors/temp` ✓ `area1/zone2/light`
Root wildcard	`#`	Matches *everything* in broker. (Use with caution - high load)

Smart Farming Example: Publisher sends `area1/humidity:65%` → Subscribers với patterns `area1/humidity`, `area1/+`, hoặc `area1/#` đều nhận được.

Quality of Service (QoS) Levels

Để xử lý network reliability khác nhau, MQTT định nghĩa 3 QoS levels:

Level	Guarantee	Mechanism & Use Case
0	At most once	Delivered once, no confirmation. Fast, có thể mất message. *Use:* Sensor readings không critical.
1	At least once	Delivered ít nhất 1 lần, requires confirmation (ACK). Có thể duplicate. *Use:* Important data.
2	Exactly once	Four-step handshake. Chậm nhất nhưng không duplicate. *Use:* Financial transactions, critical commands.

Note: Actual QoS = min(Publisher QoS, Subscriber QoS).

Practical Application: Smart Farming

- **Sensors (Publishers):**
  - *DHT11:* Temperature & air humidity.
  - *BH1750:* Luminosity (độ sáng).
  - *DHT22:* Soil moisture (độ ẩm đất).
- **Flow:** Sensors publish data đến broker với topics như `area1/moisture` → Management system subscribes topics → Real-time monitoring & decision making.
- **Benefits:** Cost-effective, scalable, knowledge base creation cho farming management.

3.9.2 MQTT vs HTTP Comparison

Aspect	HTTP	MQTT
Model	Request-Response (Client-Server)	Publish-Subscribe (qua Broker)
Connection	Short-lived (đóng sau request)	Persistent (Keep-alive, lightweight)
Header Size	Large (KB range) Cồng kềnh metadata	Very small (2 bytes) Tiết kiệm bandwidth
Power	High consumption	Low power (tối ưu battery devices)
Network	Reliable networks	Unreliable networks (chập chờn, high latency)
Use Cases	• Web browsing • REST APIs • Large file transfers	• IoT sensors • M2M communication • Constrained devices

Key Insight: HTTP designed cho human-to-machine (web), MQTT designed cho machine-to-machine (IoT).

3.10 Pipelines & Orchestration

3.10.1 Apache Kafka vs Airflow

Apache Kafka	Apache Airflow
Loại: Event Streaming Platform (Message Broker).	**Loại:** Workflow Orchestration (Quản lý quy trình).
Đặc trưng: • Log bền vững (Durable log). • Decoupling (Tách rời). • Replayable, High throughput.	**Đặc trưng:** • Code-as-infra (Python DAGs). • Quản lý dependency phức tạp. • Backfill (chạy lại quá khứ).
Vai trò: "Xương sống"vận chuyển dữ liệu *Real-time*.	**Vai trò:** "Nhạc trưởng"điều phối Job (*Batch/ETL*).

3.10.2 CDC (Change Data Capture)

Theo dõi và đồng bộ thay đổi từ database nguồn sang đích.

- **Mục đích:** Real-time data replication, sync giữa OLTP và OLAP, event-driven architecture.
- **Cơ chế:**
  - *Log-based CDC:* Đọc database transaction log (binlog MySQL, WAL PostgreSQL). **Tốt nhất** - không ảnh hưởng source.
  - *Trigger-based:* Trigger trên INSERT/UPDATE/DELETE. Ảnh hưởng performance.
  - *Timestamp/Version-based:* Poll dựa trên `updated_at` column. Thiếu DELETE events.
- **Tools:** Debezium (Kafka Connect), AWS DMS, Oracle GoldenGate, Airbyte.
- **Use Case:** Sync OLTP → DW, Microservices data sharing, Cache invalidation, Audit logs.
- **Pattern:** Source DB → CDC Tool → Kafka → Sink (DW/Cache/Search).

3.10.3 ETL vs ELT

- **ETL (Extract-Transform-Load):** Transform *trước* khi vào kho. Schema-on-write. Dữ liệu sạch, bảo mật. (Truyền thống).
- **ELT (Extract-Load-Transform):** Load raw vào kho *trước*, transform sau. Schema-on-read. Tận dụng sức mạnh Cloud DW (BigQuery, Snowflake). (Hiện đại).
- **Bản chất chung:** Cả **ETL** và **ELT** đều là các quá trình hỗ trợ **tích hợp dữ liệu** (data integration): trích xuất dữ liệu từ nguồn, đưa vào đích (DW/Sink) và chuẩn hoá/chuẩn bị dữ liệu cho phân tích hoặc hệ thống downstream.
- **Ngữ cảnh sử dụng:**
  - **ELT:** Hiệu quả với dữ liệu phức tạp hoặc khối lượng rất lớn (Big Data), khi kho dữ liệu/Cloud DW có khả năng xử lý transform ở quy mô (ví dụ: BigQuery, Snowflake).
  - **ETL:** Phù hợp với các ngữ cảnh cơ sở dữ liệu truyền thống hoặc khi cần chuyển đổi, làm sạch kỹ lưỡng trước khi nạp vào kho (ví dụ do yêu cầu bảo mật, chuẩn hoá dữ liệu, hoặc tài nguyên xử lý hạn chế ở đích).
- **Lưu ý khi trả lời trong đề thi:** Quan điểm đúng nhất thường là *không* chọn tuyệt đối ETL hay ELT — cả hai đều có ưu, nhược điểm riêng và việc lựa chọn phụ thuộc vào ngữ cảnh ứng dụng cụ thể (yêu cầu hiệu năng, kích thước dữ liệu, khả năng xử lý của DW, chính sách bảo mật, v.v.).

Common ETL Logic Patterns

Các pattern phổ biến khi thiết kế ETL flow (cho luận văn/essay).

- **Soft Deletes:** Nếu source có cờ `is_deleted`, lọc bỏ trong Extract, hoặc load vào DW với `Status = 'Inactive'` trong dimension (không xóa row khỏi Fact table).
- **Date Derivation:** Luôn mở rộng cột `Date` đơn giản thành **Date Dimension** đầy đủ (Day, Week, Month, Quarter, Year, HolidayFlag).
- **Calculated Measures:** Tính toán như `Revenue = Price * Quantity` trong giai đoạn **Transform**, lưu vào Fact table để aggregation nhanh.
- **SCD for Price Changes:** Khi `Item_price` thay đổi, dùng **Type 2 SCD** trong dimension hoặc snapshot fact table để lưu lịch sử giá.

3.11 Kho Dữ liệu (Data Warehousing)

3.11.1 OLTP vs OLAP

	OLTP (Transactional)	OLAP (Analytical)
Mục tiêu	Vận hành hàng ngày (Operational).	Ra quyết định (Decision support).
Dữ liệu	Hiện hành, chi tiết, cập nhật liên tục.	Lịch sử, tổng hợp, đa chiều.
Truy vấn	Đơn giản, trả về ít dòng (Lookup).	Phức tạp, join nhiều, quét bảng lớn.
Thiết kế	Chuẩn hóa cao (3NF) để tránh dị thường.	Phi chuẩn hóa (Star/Snowflake) để đọc nhanh.
User	NV, App, Khách hàng.	Manager, Data Analyst.

OLAP Data Cube Operations *Thao tác trên khối dữ liệu đa chiều (Dimensions: Product, Time, Location, ...).*

- **Slice:** Cố định 1 dimension, xem ma trận 2D còn lại.
  VD: Slice Year=2024 → Xem doanh thu theo (City, Product) cho năm 2024.
- **Dice:** Chọn sub-cube với nhiều điều kiện trên nhiều dimensions.
  VD: Dice (Year ∈ [2023,2024], City ∈ [HCM, HN]) → Sub-cube 2x2.
- **Roll-up (Drill-up):** Tổng hợp lên mức cao hơn (chi tiết → tổng quát).
  VD: Roll-up từ City → Region → Country.
- **Drill-down:** Phân rã xuống mức chi tiết hơn (tổng quát → chi tiết).
  VD: Drill-down từ Year → Quarter → Month → Day.
- **Pivot (Rotate):** Xoay trục để thay đổi góc nhìn.
  VD: Đổi trục X từ Product sang Time.

3.11.2 4 Đặc trưng Chính (Inmon)

Top-down, mục đích hỗ trợ ra quyết định cho quản lý

1. **Hướng chủ đề (Subject-oriented):** Tổ chức theo chủ đề chính (Khách hàng, Sản phẩm) thay vì theo ứng dụng (App Bán hàng, App Kho).
2. **Tích hợp (Integrated):** Dữ liệu từ nhiều nguồn được làm sạch, đồng nhất (đơn vị, format, encoding) trước khi nạp.
3. **Bất biến (Non-volatile):** Dữ liệu đã vào DW thì (thường) không bị sửa/xóa, chỉ đọc.
4. **Biến thiên theo thời gian (Time-variant):** Mọi dữ liệu đều gắn với mốc thời gian để phân tích xu hướng (Historical data).

3.11.3 Thách thức Xây dựng DW

- **Data Quality:** "Garbage In, Garbage Out". Dữ liệu nguồn bẩn làm sai lệch báo cáo.
- **ETL Complexity:** Tích hợp các hệ thống cũ (Legacy) rất phức tạp.
- **Performance:** Truy vấn phân tích tốn tài nguyên, cần tối ưu index/partition.

– **User Acceptance:** Người dùng không hiểu hoặc không tin tưởng dữ liệu.
– **Cost:** Chi phí lưu trữ và duy trì hạ tầng cao.

3.11.4 Mô hình hóa (Modeling)

– **Star Schema:** Fact ở giữa, Dimension xung quanh. Phi chuẩn hóa dimension. *Hiệu năng cao, dễ query.*
– **Snowflake Schema:** Chuẩn hóa dimension (tách nhỏ). *Tiết kiệm không gian, join phức tạp.*

Fact Table Additivity
Phân loại measures theo khả năng aggregation.

Type	Definition	Example	Rule
Additive	SUM qua **tất cả** dimensions.	Sales_Amount, Quantity	SUM over Time? Yes.
Semi-Add	SUM qua **một số** dims, **trừ** Time.	Account_Balance, Inventory	SUM over Time? **No**.
Non-Add	**Không** thể SUM qua bất kỳ dim.	Unit_Price, Margin_Rate	Dùng AVG, MIN, MAX.

3.11.5 Kimball's 4-Step Dimensional Modeling

Phương pháp thiết kế DW theo Kimball (bottom-up, theo chủ đề nghiệp vụ).

1. **Select the Business Process:** Chọn quy trình nghiệp vụ cần phân tích.
   VD: Orders, Billing, Inventory, Shipments.
2. **Declare the Grain:** Xác định mức chi tiết của mỗi row trong fact table.
   VD: One row per order line item, one row per transaction.
   Lưu ý: Phải xác định grain **TRƯỚC** khi chọn dimensions/facts.
3. **Identify the Dimensions:** Xác định các chiều phân tích (Who, What, Where, When).
   VD: dim_customer, dim_product, dim_store, dim_date.
4. **Identify the Facts:** Xác định các measures/metrics (Số liệu đo lường).
   VD: quantity_sold, revenue, discount_amount, cost.

3.11.6 Schema Evolution Trade-offs

Khi yêu cầu thay đổi, có 2 chiến lược phổ biến để mở rộng schema DW.

– **Strategy 1: Extend Star Schema (Add Columns to Dimension)**
  – *Pros:* Queries đơn giản hơn (không join mới), backward compatible.
  – *Cons:* Dimension table trở nên rất rộng (wide/fat dimension), tiềm ẩn redundancy và anomalies.
– **Strategy 2: Convert to Snowflake (Normalize New Attributes)**
  – *Pros:* Tiết kiệm storage, cấu trúc rõ ràng hơn cho dữ liệu phân cấp (VD: Region → City).
  – *Cons:* Breaking Change - queries cũ cần sửa (thêm join mới), ETL phức tạp hơn (quản lý keys giữa các bảng normalized).

3.11.7 DW vs DL vs Lakehouse

– **Data Warehouse:** Dữ liệu có cấu trúc, cho BI/Reporting.
– **Data Lake:** Dữ liệu thô (Raw), đa dạng, giá rẻ, cho ML/DS.
– **Lakehouse:** Kết hợp (Lưu trữ rẻ của Lake + Quản lý/ACID của Warehouse).

3.11.8 SCD (Slowly Changing Dimensions)

Tại sao cần? Để đảm bảo báo cáo lịch sử chính xác. Nếu KH chuyển từ HCM ra HN năm 2024, doanh số năm 2020 vẫn phải tính cho HCM.

Type	Chiến lược	Đặc điểm & Use Case
0	**Retain Original:** Giữ nguyên, không bao giờ sửa.	Dữ liệu gốc là chân lý. (VD: Ngày sinh).
1	**Overwrite:** Ghi đè giá trị mới lên cũ.	Không cần lịch sử. Sửa lỗi chính tả.
2	**Add Row:** Add new row + *Effective Date* + *Current_Flag*.	**Most standard**. Track complete history of changes.
3	**Add Column:** Thêm cột *Previous_Value*.	Chỉ cần biết giá trị liền trước. (Ít dùng).
4	**Add History Table:** Tách bảng lịch sử riêng (Mini-Dimension).	Tối ưu khi bảng chính quá lớn và chỉ một nhóm thuộc tính thay đổi nhanh.
5	**Hybrid (4 + 1):** Mini-dimension + "Current"reference in main table.	Optimize queries when need both detailed history and current value quickly.
6	**Hybrid (1 + 2 + 3):** Type 2 row + column containing current value (Type 1).	*"Pure Type 6"*: Helps query history while still easily group by current value.
7	**Hybrid (Dual Keys):** Fact table contains both *Surrogate Key* (history) and *Natural Key* (current).	Most flexible: Join by Surrogate to view history, join by Natural to view current.

Các kỹ thuật xử lý dữ liệu thay đổi theo thời gian (0 → 7).

3.12 Quản lý Dữ liệu

3.12.1 Vấn đề Tích hợp (Data Integration Issues)

– **Heterogeneous data sources:** Khác biệt về hệ quản trị cơ sở dữ liệu và định dạng tệp.
  VD: MySQL, PostgreSQL, Oracle, MongoDB, CSV, JSON, XML, Parquet.
– **Data Mapping:** Khác biệt về cấu trúc schema giữa các nguồn.
  VD: Bảng 'Employee' (Full_Name, DOB) vs 'Emp' (First_Name, Last_Name, Birth_Date).
– **Data Conflicts:** Conflicts in data type, value, format, unit, precision.
  Type: String vs Integer for employee code. *Format:* DD/MM/YYYY vs MM/DD/YYYY.
  Unit: USD vs VND, km vs miles. *Precision:* 2 vs 4 decimal places.
– **Data Redundancy:** Dữ liệu trùng lặp từ nhiều nguồn cần khử trùng (Deduplication).
  VD: Cùng khách hàng xuất hiện trong CRM và ERP với ID khác nhau.
– **Entity Resolution:** Xác định 2 bản ghi từ 2 nguồn khác nhau là cùng 1 thực thể.
  VD: 'Nguyen Van A' (DB1) và 'A Nguyen' (DB2), 'IBM' vs 'International Business Machines'.
  Kỹ thuật: Fuzzy matching, similarity scores (Levenshtein distance), Master Data Management (MDM).
– **Constraints Violation:** Constraint violations during data integration.
  Primary Key: Duplicate primary keys during merge. *Foreign Key:* References don't exist.
  Semantic: Invalid values (negative age, future dates).
– **Data Quality Issues:** Data quality problems during integration process.
  Accuracy: Deviates from reality. *Completeness:* Missing required fields (NULL).
  Uniqueness: Duplicates. *Timeliness:* Stale/outdated data. *Consistency:* Conflicts between sources.
– **Communication Heterogeneity:** Khác biệt về giao diện và giao thức truyền thông.
  VD: REST API vs SOAP, HTTP vs FTP, Batch files vs Real-time streams, GraphQL vs SQL.

3.12.2 Data Quality Dimensions

Các tiêu chí đánh giá chất lượng dữ liệu trong hệ thống thông tin.

6 Core Dimensions

1. **Accuracy (Chính xác):** Mức độ dữ liệu phản ánh đúng đối tượng/sự kiện thực tế.
   Định nghĩa: Sự không chính xác có nghĩa là hệ thống biểu diễn một trạng thái thế giới thực khác với trạng thái đáng lẽ phải được biểu diễn.
   VD: Địa chỉ sai, số điện thoại cũ, thông tin lỗi thời.
2. **Completeness (Đầy đủ):** Tỷ lệ dữ liệu được lưu trữ so với khả năng 100% hoàn chỉnh.
   Định nghĩa: Khả năng của hệ thống biểu diễn mọi trạng thái có nghĩa của hệ thống thế giới thực.
   VD: NULL values, missing fields, incomplete records (missing email, phone number).
3. **Consistency (Nhất quán):** Sự vắng mặt của sự khác biệt khi so sánh hai hoặc nhiều biểu diễn của cùng một thứ.
   Định nghĩa: Sự không nhất quán có nghĩa là ánh xạ biểu diễn là một-nhiều (one-to-many).
   VD: Tổng doanh thu trong bảng Orders ≠ tổng trong Report, tên KH khác nhau giữa CRM và ERP.
4. **Validity (Hợp lệ):** Dữ liệu hợp lệ nếu tuân thủ cú pháp của định nghĩa (format, type, range).
   VD: Email correct format (has @), age in range [0, 150], enum has allowed values.
5. **Timeliness (Kịp thời):** Mức độ dữ liệu phản ánh thực tế từ mốc thời gian yêu cầu.
   Định nghĩa: Độ trễ giữa thay đổi trạng thái thế giới thực và sửa đổi tương ứng trong hệ thống.
   VD: Giá cổ phiếu cập nhật delay 10s, báo cáo tồn kho cách 1 ngày (stale data).
6. **Uniqueness (Duy nhất):** Không có thứ nào được ghi lại nhiều hơn một lần dựa trên cách xác định thứ đó.
   VD: Không có duplicate records, một khách hàng chỉ có một ID duy nhất.

Extended Dimensions

- **Interpretability (Khả năng Diễn giải):** Liên quan đến tài liệu và metadata có sẵn để diễn giải chính xác ý nghĩa và thuộc tính của nguồn dữ liệu.
  VD: Data dictionary, schema documentation, column descriptions, business glossary.
- **Accessibility (Khả năng Truy cập):** Đo lường khả năng người dùng truy cập dữ liệu từ văn hóa, trạng thái/chức năng vật lý và công nghệ có sẵn của họ.
  VD: API with rate limit, role-based access (RBAC), multi-language support.
- **Usability (Khả dụng):** Đo lường hiệu quả, hiệu suất, sự hài lòng mà người dùng cụ thể cảm nhận và sử dụng dữ liệu.
  VD: Dashboard dễ hiểu, query response time nhanh, format phù hợp với use case.
- **Trustworthiness:** Đo lường mức độ đáng tin cậy của tổ chức trong việc cung cấp nguồn dữ liệu.
  VD: Clear data lineage, complete audit logs, data provenance.

3.12.3 Quản lý Thông tin

Quản lý dữ liệu như một tài sản chiến lược của doanh nghiệp.

6 Khía cạnh Quản lý Thông tin

1. **Information Collection (Thu thập):** Xác định và thu thập dữ liệu từ các nguồn khác nhau.
   Hoạt động: Data ingestion, ETL pipelines, API integration, web scraping, sensors/IoT.
   VD: Crawl web data, stream from Kafka, batch import from CSV, CDC from transactional DB.
2. **Information Organization (Tổ chức):** Cấu trúc hóa và phân loại dữ liệu để dễ quản lý và truy xuất.
   Hoạt động: Schema design, data modeling (Star/Snowflake), taxonomy, metadata management.
   VD: Design DW dimensions/facts, create data catalog, tag data by domain.
3. **Information Storage (Lưu trữ):** Chọn và triển khai hệ thống lưu trữ phù hợp với đặc điểm dữ liệu.
   Hoạt động: Choose DBMS (RDBMS, NoSQL, DW), partitioning, replication, backup strategy.
   VD: OLTP on PostgreSQL, OLAP on Snowflake, unstructured data on S3/Data Lake.
4. **Information Manipulation (Thao tác):** Thực hiện các thao tác CRUD và chuyển đổi dữ liệu.
   Hoạt động: INSERT/UPDATE/DELETE, data transformation, cleansing, enrichment, aggregation.
   VD: Chuẩn hóa địa chỉ, deduplicate records, merge datasets, derive calculated fields.
5. **Information Processing (Xử lý):** Phân tích và tính toán để tạo insights từ dữ liệu thô.
   Hoạt động: Query execution, analytics, ML training, reporting, dashboarding, data mining.
   VD: SQL queries, Spark jobs, BI reports, predictive models, real-time analytics.
6. **Information Protection (Bảo vệ):** Đảm bảo an toàn, riêng tư và tuân thủ quy định.
   Hoạt động: Access control (RBAC), encryption (at-rest/in-transit), audit logging, compliance (GDPR).
   VD: Mã hóa PII, masking sensitive data, role-based permissions, backup/disaster recovery.

Tổng thể: Information Management bao gồm Data Governance (quản trị), Data Quality, Master Data Management (MDM), Security, và Lifecycle Management. Mục tiêu cuối cùng là đảm bảo dữ liệu **tin cậy, an toàn, dễ truy cập** để tạo giá trị kinh doanh.

3.13 Distributed Systems Challenges

*Khác với hệ thống đơn, hệ phân tán đối mặt với **partial failures** - một số phần hỏng trong khi phần khác hoạt động.*

3.13.1 8 Fallacies (Ngụy biện) của Distributed Computing

Những giả định sai lầm mà lập trình viên thường mắc phải (L. Peter Deutsch, Sun Microsystems).

1. **The Network is Reliable:** Mạng không đáng tin cậy - switches hỏng, cables ngắt, packets bị mất/reorder.
2. **Latency is Zero:** Local call (ns/μs) ≠ Remote call (ms). Chain calls → latency tích lũy.
3. **Bandwidth is Infinite:** Băng thông hữu hạn. Stamp coupling (truyền dữ liệu thừa) gây bottleneck.
4. **The Network is Secure:** Attack surface tăng. Mọi endpoint phải được bảo mật.
5. **Topology Never Changes:** Topology thay đổi liên tục (upgrades, failures, scaling) → timeout/failures.
6. **Only One Administrator:** Nhiều admin quản lý các segment khác nhau (firewall, DB) → khó phối hợp.
7. **Transport Cost is Zero:** Serialization, marshalling, network infra đều tốn chi phí.
8. **Network is Homogeneous:** Mạng gồm nhiều vendor (routers, switches) → packet loss, anomalies.

3.13.2 Unreliability: Network & Time

- **Network Unreliability:**
  - *Timeouts:* No bounded time → must use timeout. Too short: false positive. Too long: delayed detection.
  - *Network Partition (Netsplit):* Group of nodes isolated → "Split brain"(2 leaders both accept conflicting writes).
- **Time Unreliability:** "Time is an illusion không có global clock.
  - *Clock Drift:* Quartz clock drifts due to temperature. Even NTP sync has discrepancies.
  - *Process Pauses:* GC or VM suspension → node pauses longer than lease timeout → declared dead but still thinks it's alive → data corruption.
  - *Ordering Issues:* Last Write Wins (LWW) using timestamp dangerous - clocks not synced → overwrites new value with old.

3.13.3 Consistency & Consensus Challenges

- **CAP Theorem:** Can only achieve 2/3: **C**onsistency, **A**vailability, **P**artition Tolerance. Partitions unavoidable → choose CP or AP.
  → NoSQL thường chọn AP (Sẵn sàng + Chịu lỗi) thay vì CP.
- **FLP Impossibility:** Async system cannot guarantee consensus termination if 1 node crashes → must use timeout.
- **Two Generals' Problem:** Không thể đạt common knowledge (chắc chắn đồng ý) qua kênh unreliable - ACK cuối cùng có thể bị mất.
- **BASE:** **B**asically **A**vailable, **S**oft state, **E**ventual consistency (Nhất quán cuối cùng - chấp nhận dữ liệu cũ tạm thời).

3.13.4 Failure Models

- **Crash Faults:** Process stops completely, never returns (simplest).
- **Omission Faults:** Process skips steps or doesn't send/receive messages (buffer overflow, congestion).
- **Byzantine Faults:** Process behaves arbitrarily/maliciously, sends wrong messages (hardest, blockchain).

3.13.5 Managing "Truth"in Distributed Systems

- **Truth by Majority:** Không node nào tin được view của chính mình. *Sự thật = Quorum quyết định.*
  VD: Node tưởng mình là leader, nhưng quorum khai tử (do GC pause) → node đó "đã chết".
- **Fencing Tokens:** Monotonic number for storage to reject writes from nodes holding expired locks (zombie nodes).

3.13.6 Thách thức Xử lý Dữ liệu Phân tán

- **Data Skew (Lệch dữ liệu):** One partition contains too much data.
  → *Consequence:* Straggler problem - job waits for slowest node.
  → *Solution:* Salting (add random prefix) to split hot key.
- **Shuffle:** Transfer data between nodes over network (Map → Reduce).
  → *Optimization:* Broadcast Join (copy small table to all nodes) to avoid shuffling large table.

4 Appendix: Tự luận

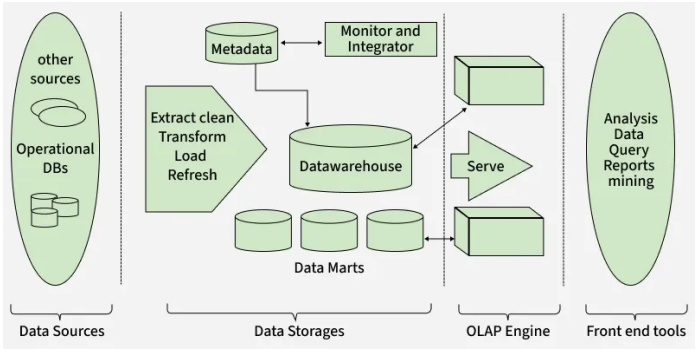
4.1 Thiết kế Kho Dữ Liệu (Star Schema)

- **Ví dụ chi tiết (Câu 35 – HK2 2022-2023 F1.1):**
  - **Nguồn (OLTP):**
    `ORDER(order_id, date, customer_name, customer_address, customer_phone, store_name, store_address, store_district, store_city, staff_id, staff_name, is_deleted)`,
    `ORDER_LINE(order_id, SKU, item_price, item_quantity, promotion_id, promotion_name, promotion_type, promotion_value)`,
    `PRODUCT(SKU, item_description, item_category, item_class, item_brand, item_cost)`.
  - **Yêu cầu:** Báo cáo **Total Revenue, Total Quantity, Total Cost** theo Date (day/week/month/year) và các chiều: Customer, Store, Staff, Product, Promotion.
- **Giải chi tiết (Kimball 4 bước):**
  1. **Business Process:** Sales.
  2. **Declare the Grain:** One row per `ORDER_LINE` (mức sản phẩm trong đơn) – cần vì đo lường theo SKU.
  3. **Identify Dimensions:**
     - **Dim_Date:** date_key, date, day, week, month, year.
     - **Dim_Customer:** customer_key, customer_name, customer_address, customer_phone.
     - **Dim_Store:** store_key, store_name, store_address, store_district, store_city.
     - **Dim_Staff:** staff_key, staff_id, staff_name.
     - **Dim_Product:** item_key, SKU, item_description, item_category, item_class, item_brand, item_cost.
     - **Dim_Promotion:** promotion_key, promotion_id, promotion_name, promotion_type, promotion_value.
  4. **Identify Facts: FACT_SALES** với FKs (date_key, customer_key, store_key, staff_key, item_key, promotion_key) và measures:
     - total_quantity = item_quantity
     - total_revenue = item_price * item_quantity
     - total_cost = item_cost * item_quantity
- **Kết quả (mô tả cấu trúc):**
  - **FACT_SALES:**

Cột	Loại	Mô tả
date_key	FK	Liên kết đến `Dim_Date`
store_key	FK	Liên kết đến `Dim_Store`
staff_key	FK	Liên kết đến `Dim_Staff`
customer_key	FK	Liên kết đến `Dim_Customer`
promotion_key	FK	Liên kết đến `Dim_Promotion`
item_key	FK	Liên kết đến `Dim_Product`
total_quantity	Measure	Tổng số lượng bán (SUM)
total_revenue	Measure	Tổng doanh thu (SUM of item_price * qty)
total_cost	Measure	Tổng chi phí (SUM of item_cost * qty)

- **ETL Note:**
  - Lọc `ORDER.is_deleted = false` khi extract để loại đơn bị hủy.
  - Tạo `Surrogate Keys` (e.g., `customer_key`) thay vì dùng PK nguồn để quản lý lịch sử và xử lý duplicate.
  - Lấy `item_cost` từ `PRODUCT` để tính `total_cost = item_cost * item_quantity` trên bước transform/load.
- **Note:** Trong bài thi, trình bày theo (1) chọn process => (2) khai báo grain (nhấn mạnh "one row per ORDER_LINE") => (3) liệt kê dimensions + attributes => (4) nêu measures và công thức tính => (5) lưu ý filter `is_deleted`.

4.2 Kiến trúc & Công nghệ



- **Mẫu giải: Thiết kế Hệ thống DWH cho Sales**
  - **Kiến trúc (3-Tier):**
    - **Source Layer:** Hệ thống OLTP (MySQL/Postgres) chứa `ORDER, ORDER_LINE, PRODUCT`.
    - **Staging Area:** Lưu trữ dữ liệu thô (Parquet trên HDFS/S3) để làm sạch và chuyển đổi.
    - **Data Warehouse Layer:** Apache Hive (HDFS) hoặc BigQuery chứa các bảng Dim/Fact theo Star Schema.
    - **Serving / Data Mart (tùy chọn):** Views / Materialized Views để phục vụ BI (PowerBI/Tableau).
  - **Công nghệ đề xuất:**

Thành phần	Công nghệ	Lý do / Ghi chú
Source	MySQL / PostgreSQL	RDBMS OLTP phổ biến, ACID đảm bảo tính nhất quán.
ETL/ELT	Apache Spark/Kafka	Xử lý Batch & Streaming, in-memory, dễ scale.
Orchestration	Apache Airflow	Lên lịch, quản lý workflow, retry, monitoring.
Storage (DWH)	Apache Hive (HDFS) / BigQuery	Hive phù hợp on-prem/Hadoop; BigQuery phù hợp cloud/serverless.
Presentation	PowerBI / Tableau	Kết nối trực tiếp tới DWH cho dashboard báo cáo.

  - **Quy trình tích hợp dữ liệu (ETL/ELT):**
    1. **Extract:**
       - Tần suất: Daily (Incremental), ví dụ: trích xuất các `ORDER` của ngày hôm qua.
       - Chỉ lấy `ORDER.is_deleted = false` để không tính các đơn hủy.
       - Công cụ: Spark đọc từ MySQL/Postgres và ghi Parquet tới Staging (HDFS/S3).
    2. **Transform:**
       - Làm sạch: Chuẩn hoá ngày, loại bỏ NULL/invalid, validate dữ liệu.
       - Tính toán measures: `revenue = item_price * item_quantity`, `cost = item_cost * item_quantity`.
       - Tra cứu & ánh xạ Surrogate Keys: map `customer_id` -> `customer_key` v.v.
       - Xử lý SCD (Type 1/Type 2) cho dimension khi cần.
    3. **Load:**
       - Ghi Fact/Dimensions vào Hive (hoặc load vào BigQuery) theo Star Schema.
       - Chiến lược nạp: Fact incremental, Dimension upsert / SCD policy.
       - Toàn bộ workflow được điều phối bởi Airflow (schedule, dependencies, retries).
  - **Mô tả tóm tắt (dùng để trả lời câu tự luận / vẽ hình):**
    Hệ thống sử dụng kiến trúc **3-Tier**. Dữ liệu bán hàng từ MySQL được trích xuất hàng ngày bằng Apache Spark, lưu tạm tại HDFS (Staging) dưới dạng Parquet. Spark

thực hiện làm sạch, tính toán doanh thu/chi phí và ánh xạ Surrogate Keys, sau đó nạp vào Apache Hive (Data Warehouse) theo Star Schema. Toàn bộ quy trình được điều phối bởi Apache Airflow và dashboard được hiển thị qua PowerBI/Tableau.

- **Ghi chú:**
  - Nêu rõ tần suất (daily), policy lọc (is_deleted), SCD policy, và nơi lưu Staging (HDFS/S3).
  - Nếu đề cho phép Cloud, thay Hive bằng BigQuery và nhấn mạnh ưu điểm serverless/fast queries.

4.3 SQL & ETL Logic

- **Extract (Trích xuất):**
  - *Filter:* Lọc data rác/cũ (VD: `WHERE is_deleted = false`).
  - *Dimensions:* Dùng `SELECT DISTINCT` từ bảng transaction để lấy list duy nhất (Khách hàng, SP).
  - *Facts:* `JOIN` các bảng giao dịch (Order + OrderDetail) để lấy đủ thông tin.
- **Transform (Biến đổi):**
  - *Calc Measures:* Tính cột phái sinh (VD: `Revenue = Price * Quantity`).
  - *Time Hierarchy:* Tách `Date` → Day, Month, Quarter, Year.
  - *Loc Hierarchy:* Tách Address → District, City, Region.
  - *Lookups:* Map ID từ nguồn sang Surrogate Key của DW.
- **Load (Nạp):**
  - *Fact:* Nạp **Incremental** (chỉ nạp dòng mới).
  - *Dimension:* Nạp **Overwrite** (nếu nhỏ) hoặc **SCD/Upsert** (nếu lớn/cần lịch sử).
  - *Schedule:* Định thời chạy (Daily/Weekly).

4.4 Lý thuyết Big Data

- **CAP vs ACID:**
  - **CAP:**
    1. **C — Consistency:** Replica consistency — tất cả bản sao nhìn thấy cùng dữ liệu tại cùng mốc.
    2. **A — Availability:** Mọi nút đang hoạt động trả lời yêu cầu (không im lặng).
    3. **P — Partition Tolerance:** Hệ thống tiếp tục hoạt động dù network partition.
  - **Khác biệt chính so với ACID:**
    - **Phạm vi:** CAP = hệ thống phân tán; ACID = giao tác (transaction).
    - **Ý nghĩa của "C":** CAP-C = Replica consistency; ACID-C = Logical integrity/constraints.
    - **NOTE:** Viết 3 định nghĩa C/A/P, nêu scope và khác biệt của "C"; ví dụ: Cassandra/Dynamo (CAP trade-offs), Postgres/MySQL (ACID).

5 Appendix: Trắc Nghiệm

5.1 Thiết kế CSDL (ERD & Normalization)

Q: Đặc tính BẮT BUỘC khi phân rã lược đồ? → **Bảo toàn thông tin (Lossless Join).**
Bẫy: "Bảo toàn phụ thuộc"chỉ là *mong muốn*, không bắt buộc.

Q: Fan Trap xảy ra khi nào? → Một thực thể liên kết 1:N với 2 thực thể khác tạo ra đường dẫn mơ hồ.
VD: NhanVien (N) ← (1) PhongBan (1) → (N) DuAn. Không biết NV nào làm DA nào.
Bẫy: Nhầm với **Chasm Trap** (thiếu đường dẫn do partial participation).

Q: Dimension trong Star Schema có chuẩn hóa không? → **KHÔNG.** Dimension thường denormalized để giảm JOIN.
Bẫy: Nếu Dimension được chuẩn hóa → đó là **Snowflake Schema**.

Q: Chọn DBMS ở giai đoạn nào? → **Trước/Bắt đầu Thiết kế Logic** (vì logic design phụ thuộc mô hình DBMS).

Q: Lược đồ ý niệm (Conceptual Schema) đặc điểm? → **Độc lập công nghệ**, dễ đọc hiểu, dùng cho cả SQL và NoSQL.
Bẫy: "Khó đọc, cần tài liệu mô tả"là SAI.

Q: Category (Union) khác Specialization/Generalization?
→ Category **KHÔNG kế thừa tất cả** thuộc tính từ các superclass. Chỉ **đại diện** cho một trong các superclass.
VD: Person UNION Company → Owner (chỉ là Person HOẶC Company, không phải cả hai).

Q: Ánh xạ (E)ERD sang Relational đảm bảo gì? → Đảm bảo: **Primary Key, Foreign Key, Participation, Cardinality**.
Lưu ý: Tất cả ràng buộc từ ERD được chuyển sang SQL constraints.

Q: Weak Entity được ánh xạ thế nào? → Tạo bảng riêng với **composite PK = (Owner FK + Partial Key)**.
Bẫy: Nghĩ weak entity không cần bảng riêng.

Q: Kiểu thực thể Shared Subclass? → Kế thừa **TẤT CẢ** thuộc tính và mối liên kết từ **NHIỀU** superclass.
VD: Student_Employee kế thừa từ cả Student VÀ Employee.

Q: Lỗi kỳ dị dữ liệu (Anomaly) gồm? → **Insertion, Update, Deletion Anomaly**.
VD: Không thể thêm Department mới nếu chưa có Employee (Insertion).

Q: Tìm Candidate Key? → Tập thuộc tính nhỏ nhất xác định tất cả thuộc tính khác (không thể bỏ bớt).
Công thức: Tính closure X^+, nếu X^+ = All Attributes → X là superkey.

5.2 Chỉ mục (Indexing & Physical Design)

Q: Số lượng Index Entries của Clustering Index? → Bằng số **Distinct Values** của trường đánh index.
Bẫy: Nhầm thành "Số Records"(Secondary) hoặc "Số Blocks"(Primary).

Q: Root của B-Tree chiếm bao nhiêu block? → Chính xác **1 Disk Block**.
Bẫy: Khi root đầy → split và tăng height, nhưng root mới vẫn chỉ 1 block.

Q: Tại sao B+-Tree tốt cho Range Query? → Vì leaf nodes được liên kết (Linked List). Chỉ cần tìm điểm đầu và quét ngang.
Bẫy: Chọn "Vì chiều cao thấp hơn". Đúng nhưng không phải lý do chính.

Q: `IS NOT NULL` có dùng Index không? → Thường **KHÔNG**. B-Tree tiêu chuẩn không index NULL → Full Table Scan.

Q: Chi phí truy cập B+-Tree: 3 hay 4 blocks? (Height=3)
→ **4 Blocks** (Nếu Unclustered).
TH 3 blocks: **Clustered Index** (Data nằm ngay tại Leaf).
TH 4 blocks: **Unclustered/Secondary Index** (Leaf chứa Pointer → tốn thêm 1 I/O để đọc Data Block).
Mẹo: Nếu đề không nói rõ, thường giả định là **Secondary Index** (Unclustered) → chọn **4**.

Q: Primary Index số entry? → Bằng **số blocks** của data file (1 entry/block đầu tiên).
Công thức: $\lceil \frac{\text{Số Records}}{\text{bfr (blocking factor)}} \rceil$.

Q: Secondary Index trên Key Field là Dense hay Sparse? → Luôn **DENSE** (mỗi record 1 entry).
Bẫy: "Có thể sparse"là SAI. Secondary trên key phải dense.

Q: Multilevel Index dùng cho loại nào? → Có thể dùng cho **TẤT CẢ** (Primary, Clustering, Secondary).
Mục đích: Giảm số block accesses khi index quá lớn.

Q: Hash Index tốt cho query nào? → **Equality (=)** tuyệt vời. **Range (<, >, BETWEEN)** tệ (phải full scan).
Khi nào dùng: WHERE code = 10 → Hash. WHERE code > 10 → B+-Tree.

Q: Bitmap Index tốt khi nào? → Trường có **cardinality thấp** (ít giá trị distinct).
VD: Gender (M/F), Status (Active/Inactive). *Tệ cho:* SSN, Email.

Q: Index trên `lower(Name)`? → Tạo **Function-Based Index**: `CREATE INDEX idx ON T(lower(Name))`.
Lưu ý: Index thường `(Name)` không dùng được cho `WHERE lower(Name)=...`.

Q: Composite Index (A, B) hỗ trợ query nào? → **WHERE A=... [AND B=...]** OK. **WHERE B=...** KHÔNG dùng được.
Nguyên tắc: Phải có điều kiện trên **cột đầu tiên** mới dùng được index.

Q: Unspanned vs Spanned storage? → **Unspanned:** 1 record không nằm trên >1 block (dễ quản lý, lãng phí không gian).
Spanned: Record có thể nằm trên nhiều block (tiết kiệm không gian, phức tạp hơn).

Q: Thiết kế Physical cần chú ý gì? → Kiểu dữ liệu, kích thước, ràng buộc, index, partitioning.
Bẫy: "Dùng String cho Date"→ SAI (nên dùng DATE type để validate).

5.3 Big Data & Data Warehousing

Q: "C"trong CAP vs "C"trong ACID khác nhau thế nào?
→ **CAP-C:** Consistency of Replicas (mọi node cùng nhìn thấy dữ liệu).
→ **ACID-C:** Data thỏa ràng buộc toàn vẹn (Constraints).
Bẫy: Đánh đồng hai chữ "Consistency".

Q: MapReduce lưu dữ liệu trung gian ở đâu? → Ghi xuống **Đĩa cứng (Disk)** của local node.
Bẫy: Nhầm với **Spark**, Spark cache trên **RAM** (nhanh hơn).

Q: Thứ tự đúng của Kimball 4-Step? → (1) Business Process → (2) **Grain** → (3) Dimensions → (4) Facts.
Bẫy: Xác định Dimensions/Facts trước **Grain** → toàn bộ thiết kế sai.

Q: Fact Table chứa gì? → FK trỏ Dimensions + **Measures (số đo)**.
Bẫy: Nghĩ Fact chứa thông tin mô tả. Fact thường toàn số và ID.

Q: Mục đích Data Warehouse theo Inmon? → **Hỗ trợ ra quyết định** (Decision Support).
4 đặc điểm: Subject-oriented, Integrated, Time-variant, Non-volatile.

Q: Subject-oriented nghĩa là gì? → Dữ liệu xoay quanh **chủ đề nghiệp vụ** (khách hàng, sản phẩm, doanh thu) chứ không phải giao dịch.

Q: ETL vs ELT khác nhau? → **ETL:** Transform trước khi Load (truyền thống, data nhỏ/vừa).
ELT: Load rồi Transform tại đích (Big Data, cloud, dữ liệu phức tạp).

Q: Star Schema vs Snowflake Schema? → **Star:** Dimension denormalized (ít JOIN, dễ query).
Snowflake: Dimension normalized (ít redundancy, nhiều JOIN).
Lưu ý: **Fact table LUÔN normalized** trong cả hai.

Q: Fact vs Dimension relationship? → **One-to-Many** (1 Dimension → N Facts).
VD: 1 Product có thể bán nhiều lần (nhiều dòng trong Fact).

Q: Additive, Semi-Additive, Non-Additive Measure? → **Additive:** Cộng được trên mọi dimension (Revenue, Quantity).
Semi-Additive: Cộng được trên 1 số dimension (Balance - không cộng theo time).
Non-Additive: Không cộng được (Ratio, Average, Price).

Q: Grain (độ mịn dữ liệu) là gì? → **Mức chi tiết thấp nhất** của 1 dòng trong Fact.
VD: Grain = "Mỗi dòng là 1 giao dịch bán hàng của 1 khách tại 1 thời điểm".

Q: Hierarchy trong Dimension? → Cấu trúc phân cấp để drill-down/roll-up.
VD: Date: Day → Week → Month → Quarter → Year.

Q: Data Warehouse vs Data Mart? → **DW:** Toàn doanh nghiệp, nhiều subject.
Data Mart: 1 bộ phận/chủ đề cụ thể (Sales DM, HR DM).

Q: Data Lake đặc điểm? → Lưu **raw data**, mọi cấu trúc, **schema-on-read**.
Khác DW: DW cần transform trước (schema-on-write).

Q: Data Lakehouse là gì? → Kết hợp **flexibility của Lake + structure của Warehouse**.
Đặc điểm: Quản lý metadata/index thống nhất, hỗ trợ cả structured và unstructured.

Q: OLTP vs OLAP? → **OLTP:** Giao dịch hàng ngày (INSERT/UPDATE nhiều), dữ liệu hiện tại.
OLAP: Phân tích (SELECT phức tạp), dữ liệu lịch sử, read-heavy.

5.4 Big Data Technologies

Q: 5V của Big Data? → **Volume, Velocity, Variety, Veracity, Value**.
Ghi nhớ: Lượng lớn, Nhanh, Đa dạng, Chất lượng, Giá trị.

Q: HDFS hoạt động thế nào? → **NameNode:** Lưu metadata.
DataNode: Lưu data blocks (replication mặc định = 3).
Fault tolerance: Nếu 1 node chết, dùng replica ở node khác.

Q: MapReduce hoạt động? → **Map:** (key, value) → intermediate (key, list of values).
Shuffle: Group by key. **Reduce:** Aggregate từng group.
Lưu ý: Dữ liệu trung gian ghi xuống **disk**.

Q: Spark khác Hadoop MapReduce? → Spark dùng **RDD in-memory** (nhanh hơn 10-100x).
MapReduce ghi disk mỗi stage. Spark có thể chain operations.

Q: RDD trong Spark là gì? → **Resilient Distributed Dataset** - tập dữ liệu phân tán, immutable, fault-tolerant.
Fault recovery: Dùng **lineage** (ghi lại cách tính) để recompute.

Q: Batch vs Stream Processing? → **Batch:** Xử lý lô lớn định kỳ (hourly/daily).
Stream: Xử lý real-time/near real-time (seconds).
VD Stream: Fraud detection, IoT sensors, stock trading.

Q: Spark Streaming vs Kafka Streams? → **Spark Streaming:** Micro-batch (giả lập stream).
Kafka Streams: True streaming (event-by-event).

Q: Checkpoint trong Spark Streaming? → Lưu **RDD + metadata** xuống HDFS để recover khi lỗi.

Q: Apache Hive là gì? → **SQL-on-Hadoop**. Viết SQL, Hive compile thành MapReduce/Spark jobs.
Use case: DW trên HDFS.

Q: CAP Theorem? → Chỉ chọn được **2 trong 3**: Consistency, Availability, Partition Tolerance.
VD: MongoDB (CP), Cassandra (AP).

Q: NoSQL models? → **Key-Value** (Redis), **Document** (MongoDB), **Column** (Cassandra), **Graph** (Neo4j).

Q: BASE vs ACID? → **BASE:** Basically Available, Soft state, Eventually consistent (NoSQL).
ACID: Atomicity, Consistency, Isolation, Durability (SQL).

Q: Master-Master vs Master-Slave? → **M-M:** Cả hai đều write được (high availability).
M-S: Master write, Slave read-only (simpler, consistency).

5.5 Query Optimization

Q: Tại sao Optimizer chuyển CROSS JOIN thành JOIN? → Để tránh tạo ra $|R| \times |S|$ dòng trung gian (rất tốn kém).
Bẫy: Nghĩa DB "không hỗ trợ"Cross Join. DB hỗ trợ nhưng Optimizer tránh.

Q: Mục tiêu tối thượng của Query Optimization? → Chọn plan có **chi phí ước tính thấp nhất** (Cost-based).
Bẫy: "Tìm plan tốt nhất tuyệt đối". Thực tế là "đủ tốt"trong thời gian cho phép.

Q: Query Rewriting là gì? → Biến đổi query thành dạng tương đương nhưng hiệu quả hơn.
VD: `WHERE A=1 AND (B=2 OR B=3)` → `WHERE A=1 AND B IN (2,3)`.

Q: Subquery vs JOIN nào nhanh hơn? → Thường **JOIN nhanh hơn** (optimizer có nhiều lựa chọn).
Lưu ý: Correlated subquery (chạy N lần) rất chậm.

Q: Selectivity là gì? → **Tỷ lệ dòng thỏa điều kiện** = $\frac{\text{matching rows}}{\text{total rows}}$.
Ứng dụng: Ước tính kích thước kết quả trung gian, quyết định thứ tự JOIN.

Q: Index được dùng khi nào? → Khi **selectivity thấp** (trả về ít dòng). Nếu trả về >15-20% → Full Scan nhanh hơn.

Q: Partition Pruning là gì? → Chỉ scan các partition liên quan, bỏ qua partition không thỏa điều kiện.
VD: `WHERE date >= '2024-01-01'` → chỉ scan partition Q1-2024.

5.6 Các Khái Niệm Khác

Q: Slowly Changing Dimension (SCD)? → Xử lý dimension thay đổi theo thời gian.
Type 1: Overwrite (mất lịch sử).
Type 2: Thêm dòng mới (valid_from, valid_to, is_active).
Type 3: Thêm cột lưu giá trị cũ.

Q: Factless Fact Table? → Fact **không có measure**, chỉ ghi nhận sự kiện xảy ra.
VD: Student-Class enrollment (chỉ cần biết ai học lớp nào, không có số đo).

Q: Surrogate Key là gì? → Khóa **nhân tạo** (auto-increment, UUID) thay natural key.
Lợi ích: Ổn định, đơn giản, hỗ trợ SCD Type 2.

Q: Junk Dimension? → Gom các **low-cardinality flags/indicators** vào 1 dimension.
VD: PaymentMethod, OrderType, ShipStatus → 1 bảng JunkDim.

Q: Degenerate Dimension? → Dimension chỉ có **1 thuộc tính** (thường là ID), không tạo bảng riêng, để luôn trong Fact.
VD: OrderID, InvoiceNumber trong Fact.

Q: Conformed Dimension? → Dimension **dùng chung** cho nhiều Fact (đảm bảo consistency).
VD: DateDim dùng cho SalesFact, InventoryFact, ShippingFact.