Distributed Databases

(NNTHIENPHUC_NOTION) Câu hỏi tổng hợp từ mình tự học + tham khảo của mấy anh chị:

▼ Lý thuyết

▼ Distributed Database: Làm gì để không bị AI thay thế?

- Xây dựng cơ sở dữ liệu, thiết kế, phân tích cơ sở dữ liệu
- Phát hiện lỗ hỏng, debug
- Phân quyền

▼ Database: 5 ràng buộc cứng trên database:

- Primary key
- Foreign key
- Unique key
- Miền giá trị
- Not null

▼ Database: Không làm được ràng buộc mềm trên database mà phải làm bằng code nhưng vẫn có thể làm trên csdl thông qua Trigger.

- Vẫn có thể viết ràng buộc mềm trên csdl → trigger
  - → làm chậm hệ thống khi chạy
  - → không cản được lỗi cơ bản ở mức client → chỉ phát huy tác dụng khi đưa vào db

→ Khi lập trình ngoài ràng buộc cứng cần phải bắt trigger trên db và bắt cả lỗi trên lập trình.

▼ Database: Dư thừa trong database vẫn xảy ra được.

▼ Distributed Database: SQL tự đồng bộ DATA 2 chiều (đẩy lên site chủ rồi truyền về site con), còn Code thì chỉ 1 chiều Site chủ → Site phụ.

▼ Distributed Database: Trên thực tế nếu ứng dụng đang viết mà chạy được trên 1 môi trường database thì đừng nên phân tán cơ sở dữ liệu bởi vì distributed database có nhiều lỗi ẩn.

▼ Distributed Database: Nếu 1 nhân viên đang làm việc ở chi nhánh này mà chuyển sang chi nhánh khác thì các hóa đơn, phiếu nhập thuộc nhân viên đó cũng sẽ bị chuyển sang ⇒ chi nhánh cũ sẽ mất doanh thu và chi nhánh mới lại tăng doanh thu mặc dù chả làm gì. Nếu chạy trên Cơ sở dữ liệu tập trung thì không sao bởi vì doanh thu chỉ nằm ở 1 chỗ.

- Giải pháp cho trường hợp này là tạo field TRANGTHAI cho nhân viên (giả định nếu đang làm việc là 0 và nghỉ việc là 1).
  - Chi nhánh cũ: TRANGTHAI là 1
  - Chi nhánh mới: Nhập thông tin của nhân viên chính xác như bên chi nhánh cũ dựa trên cccd và MaNV sẽ là mã mới và set TRANGTHAI là 0.
- Nếu nhân viên đó chuyển về chi nhánh cũ thì set TRANGTHAI bên chi nhánh cũ là 0 và chi nhánh mới là 1.

▼ Project: Vậy nếu muốn xóa vật lý thì làm sao?

- Khi bắt đầu khởi động project thì luôn phải bật LOG để ghi lịch sử hoạt động nhằm giúp dễ debug cũng như restore ⇒ Tuy chậm hơn bình thường nhưng sẽ hiệu quả hơn trong thực tế.

▼ Distributed Database: Khi cập nhật dữ liệu, tạo giao dịch liên quan đến nhiều site thì phải Tạo thao tác để giả sử khi giao dịch không thành công thì tất cả quá trình trước đó sẽ roll back.

▼ Distributed Database: Để LINK SEVER:

- Dùng giao diện:
  - IP
  - username
  - password

▼ Distributed Database: Gọi hàm từ xa nếu Môi trường là database → dùng SELECT

▼ Distributed Database: Ưu và nhược điểm khi luôn đẩy 1 mảng dữ liệu cho 2 máy khác nhau:

- Nhược điểm:
  - Tốn chi phí vì phải mua tận 2 máy
  - Tốn tg tại phải đẩy sang 2 cái
- Ưu điểm:
  - Backup dữ liệu phòng hờ 1 máy bị trục trặc thì dữ liệu vẫn nằm ở máy còn lại ⇒ tránh bị trì trệ

▼ Distributed Database: Nếu 1 quan hệ tách thành 2 mảnh (phân hoạch) thì SELECT với UPDATE?

- SELECT:
  - Chậm ← Nếu dữ liệu không nằm ở site này thì phải chuyển qua site khác mà kiếm.
  - Sever stop → Ứng dụng hết chạy.
- UPDATE:
  - Nhanh ← Chỉ update dữ liệu về site chủ.

▼ Distributed Database: Nếu 1 quan hệ nhân bản thì SELECT với UPDATE?

- SELECT:
  - Nhanh ← Dữ liệu nằm ở tất cả các site

- UPDATE:
    - Chậm ← Dữ liệu phải đưa về site chủ → site chủ phải update cho các site phân mảnh

▼ Distributed Database: 1 quan hệ ít ADD, UPDATE, DELETE ⇒ Nhân bản ⇒ Truy cập rất nhanh nhưng không an toàn vì tất cả admin sẽ nhìn thấy hết.

▼ Distributed Database: Để tránh sợ mất an toàn dữ liệu ⇒ Phân hoạch.

▼ Distributed Database: Phân mảnh ngang ⇒ UNION, Phân mảnh dọc ⇒ JOIN (có thể là INNER JOIN), Phân mảnh hỗn hợp ⇒ UNION xong rồi JOIN.

▼ Distributed Database: Luôn phải tuân thủ 3 quy tắc phân mảnh nhưng thực tế chỉ cần tuân thủ 2 cái.

▼ Project: DELETE data:

- Vật lý ⇒ Xóa mất dữ liệu ⇒ Xóa nhầm ⇒ Không thể restore ⇒ Không nên
- Luận lý ⇒ Set trạng thái (Flag) ⇒ Tránh việc mất mát data.

▼ Database: Trong quá trình phân tích thiết kế database có 3 mức:

- Mức quan niệm: Mô hình thực thể kết hợp ER
- Mức logic: Chuẩn hóa CSDL
- Mức vật lý: Thực hành. Dùng phần mềm trên máy tính (giao diện hoặc viết lệnh)

▼ Database: Có 6 loại ràng buộc toàn vẹn

Mỗi loại có 5, 6, 7,... loại (bao gồm cứng, mềm)

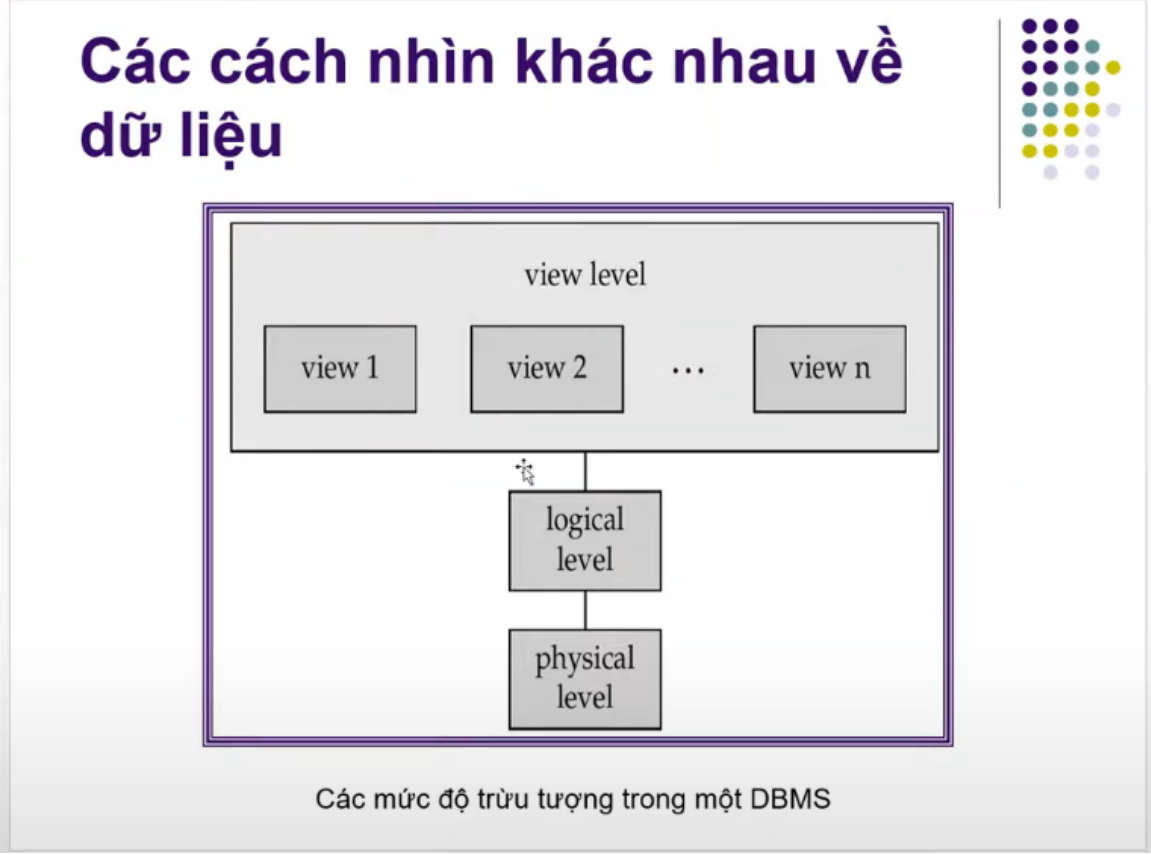
▼ Database: Khi 2 người cùng sửa trên cùng 1 mẫu tin của 1 database thì phải báo lỗi.

▼ Database: Khi cập nhật (thêm, xóa, sửa) dữ liệu gặp lỗi thì phải rollback những data đã cập nhật nhằm đảm bảo tính nhất quán ⇒ Sử dụng transaction.

▼ Database: Vấn đề khi dư thừa dữ liệu:

- Tốn bộ nhớ lưu trữ dữ liệu trùng lắp
- Gây khó khăn cho việc bảo trì
- Vấn đề lớn nhất là:
  - Khi dữ liệu thay đổi ở 1 tập tin có thể gây sự mâu thuẫn dữ liệu
  - Làm mất đi tính toàn vẹn dữ liệu

▼ Database: Các cách nhìn khác nhau về dữ liệu



▼ Database: Các loại thuộc tính:

- Simple
- Multi-valued
- Derived (Tùy trường hợp mà quyết định có nên thêm vào database không):

    - Những thuộc tính tính toán được từ 1 biểu thức đơn giản tính toán được thì không nên đưa vào: trị giá,...
    - Ngược lại: Số lượng tồn kho,...
- NULL

▼ Database: Phân biệt primary key vs unique key:

- Primary key don’t allow NULL but unique key can allow
- Both of them uniquely identifies each entity

▼ Database: Luôn thiết kế database theo góc nhìn tổng thể ⇒ Trả lời được tất cả câu hỏi

▼ Database: Các bước thiết kế 1 database:

- Tìm hiểu thực tế
- Xác định thực thể
- Vẽ ERD
- Chuyển đổi ERD sang Table

  Mỗi thực thể sẽ được chuyển thành Quan hệ thực thể.
  Mối liên kết 1-1 : quan hệ này sẽ chứa khóa chính của quan hệ kia, và ngược lại
  Mối liên kết 1-n : quan hệ đầu nhiều sẽ chứa khóa chính của quan hệ đầu 1.
  Mối liên kết n-n : sẽ có thêm quan hệ mới với các thuộc tính là khóa chính ở các thực thể, và các thuộc tính riêng của nó. Khóa chính của Quan hệ mới này sẽ là khóa tổ hợp (khóa chính của các thực thể đầu nhiều), có thể có thêm thuộc tính riêng trong khóa chính.
  Tổ hợp các quan hệ có cùng khóa chính

▼ Database: Làm sao để 1 database không bị mâu thuẫn thông tin? ⇒ Đưa về dạng chuẩn 3 (tối thiểu) (Khóa chính, Phụ thuộc hàm)

▼ Database: Khi Join 2 bảng có cần phải có FK không? Không cần phải có FK chỉ cần có field trùng nhau. Select n bảng thì cần (n-1) điều kiện kết.

▼ Database: Tại sao ngoài bắt lỗi trong csdl ta còn phải bắt lỗi trên phần mềm?

Vì khi bắt lỗi trên database người dùng nhập dữ liệu vào thì dữ liệu phải đưa xuống database để kiểm tra sau đó trả về kết quả → Tốn tg.

Giải thích rõ hơn: DB được đặt ở 1 nơi máy client đặt ở 1 nơi nên khi truy xuất về DB sẽ tốn rất nhiều thời gian → bắt lỗi trực tiếp trên máy client sẽ giảm thiểu thời gian tiêu hao.

▼ DDB: Trên site phân mảnh khi add 1 PK mới không tồn tại trên site đó nhưng tồn tại trên site chủ có được không?

Được. Nhưng sau khi đồng bộ dữ liệu lên site chủ thì dữ liệu đó tự biến mất do trên site chủ đã có PK đó rồi.

▼ DDB: Phải biết được trong đề tài chỗ nào là dùng ứng dụng phân tán hay không?

▼ DDB: Các ràng buộc cứng + mềm → Ràng buộc toàn vẹn của db tập trung đều phải đưa được vào hệ thống phân tán (tránh sự mất mát dữ liệu)

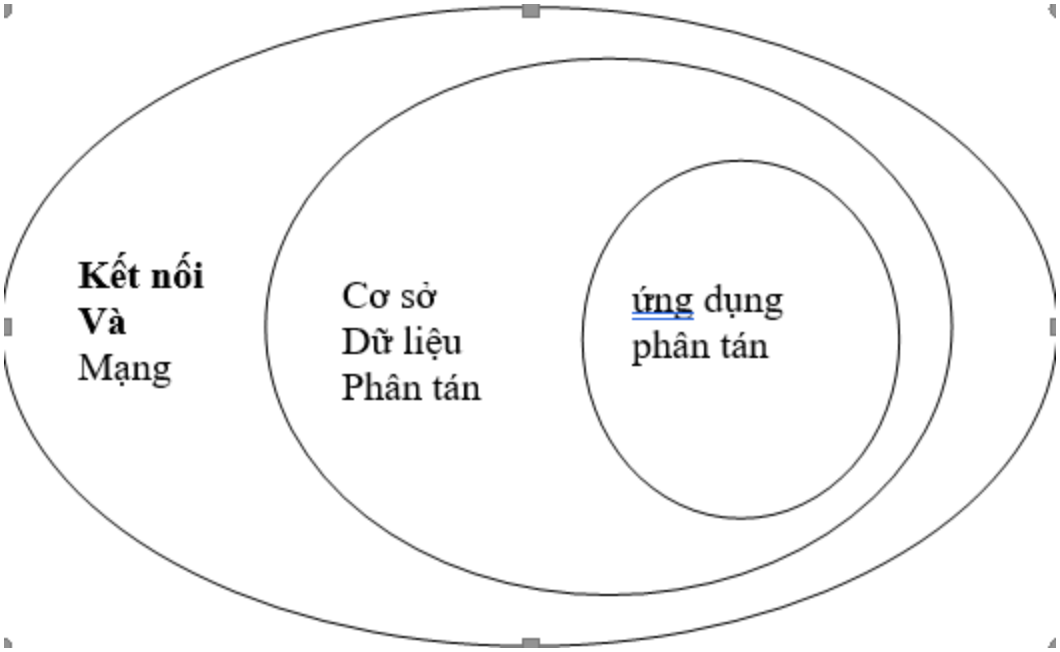
▼ DDB: Mỗi chi nhánh chỉ lưu trữ csdl cục bộ có đủ chưa? Cơ sở dữ liệu phân tán có phải là 1 tập các csdl cục bộ?

- Về mặt kỹ thuật, chúng ta thấy cần có các ứng dụng mà truy xuất dữ liệu đang đặt ở nhiều nhánh. Các ứng dụng này được gọi là ứng dụng toàn cục hay ứng dụng phân tán.
- Một ứng dụng toàn cục thông thường trong ví dụ ngân hàng là việc chuyển tiền từ 1 tk này đến tk khác. Ứng dụng này yêu cầu cập nhật csdl ở cả 2 nhánh.
- Hơn nữa ứng dụng toàn cục giúp cho người dùng không phân biệt được dữ liệu đó là cục bộ hay toàn cục. Đó là tính trong suốt dữ liệu trong csdlpt. Và đương nhiên khi ứng dụng toàn cục truy cập dữ liệu cục bộ sẽ nhanh hơn ứng dụng từ xa điều này nói lên sự nhân bản dữ liệu ở các nơi cũng làm tăng tốc độ xử lý chương trình.

▼ DDB: Định nghĩa Một CSDLPT là tập hợp dữ liệu quan hệ lẫn nhau 1 cách luận lý trên cùng 1 hệ thống nhưng được trải rộng

trên nhiều site của 1 mạng máy tính.

Mỗi vị trí có quyền tự quản cơ sở liệu cục bộ của mình và thực thi các ứng dụng cục bộ. Mỗi vị trí cũng phải ***tham gia vào việc thực thi ít nhất một ứng dụng toàn cục****: yêu cầu truy xuất dữ liệu tại nhiều vị trí qua mạng.*



▼ DDB: Cho dù mở rộng hay thu hẹp các site chương trình cũng không sửa đổi → Tự động linh hoạt → chạy đúng.

▼ DDB: Tại sao phải dùng phân tán?

- Có rất nhiều lý do nhưng chung quy lại là nhu cầu Thực tế → Tự nghiệm ra nên dùng phân tán hay tập trung.
- VD: Làm dự án chuỗi khách sạn gồm nhiều khách sạn nhỏ:
  - Tự thu tự chi
  - Mỗi khách sạn đều có quản lý và giám đốc → marketing, tiếp thị quản bá, ...
    - Nếu khách sạn nào làm ăn tốt hơn thì có lãi → nhân viên có lợi
    - Khách sạn làm không tốt → nhân viên có hại
  - Khách sạn cách nhau 1 con đường
    - 1 quá tải

    - 1 thiếu khách
    - Nhu cầu là nếu quá tải thì xem xem khách sạn gần đó mà thiếu khách không? → Điều phối khách qua → Doanh thu thì sẽ tính cho khách sạn bên kia → Chia lại % cho bên khách sạn điều phối

⇒ Bản thân từng khách sạn con toàn quyền trên db của mình nhưng vẫn thấy được db của khách sạn khác trừ khi được cấp quyền truy cập → NÊN DÙNG PHÂN TÁN.

▼ DDB: Dùng CSDLPT thì có lợi gì hay không?

- Nếu dùng DB tập trung → Phân quyền → Chỉ nhìn thấy DB của chi nhánh đó → Khuyết điểm:
    - 1 anh ở khách sạn 1 (admin) không đi qua chương trình đi thẳng vào sql có thể thấy được thông tin doanh thu của các khách sạn khác.
- Ưu điểm: Thời gian chạy ở 1 phân mảnh bé hơn tập trung. → Số lượng càng ít truy vấn càng nhanh.
    - Thông thường trên mọi hệ thống phân tán đa số là ứng dụng cục bộ (80 - 90%) và còn lại là ứng dụng phân tán.
- Về mặt đồng bộ: Phân tán lâu hơn tập trung (Dữ liệu từ phân mảnh đổ về site chủ, site chủ kiểm tra xem có site phân mảnh nào liên quan đến dữ liệu đó không → đổ về site phân mảnh có liên quan.
- Code phân tán khó hơn cục bộ

▼ DDB: So sánh các tính chất đặc trưng của CSDL tập trung và CSDL phân tán:

Tính chất đặc trưng	**Cơ sở dữ liệu tập trung**	**Cơ sở dữ liệu phân tán**
Điều khiển tập trung	- Khả năng cung cấp sự điều khiển tập trung trên các tài nguyên thông tin. - Cần có người quản trị cơ sở dữ liệu.	- Cấu trúc điều khiển phân cấp: quản trị cơ sở dữ liệu toàn cục và quản trị cơ sở dữ liệu cục bộ phân tán.

Độc lập dữ liệu	- Tổ chức dữ liệu trong suốt với các lập trình viên. Các chương trình được viết có cái nhìn "quan niệm" về dữ liệu. - Lợi điểm: các chương trình không bị ảnh hưởng bởi sự thay đổi tổ chức vật lý của dữ liệu	- Ngoài tính chất độc lập dữ liệu như trong cơ sở dữ liệu tập trung, còn có tính chất trong suốt phân tán nghĩa là các chương trình được viết như cơ sở dữ liệu không hề được phân tán.
Sự dư thừa dữ liệu	Giảm thiểu sự dư thừa dữ liệu do: - Tính nhất quán dữ liệu cao - Tiết kiệm dung lượng nhớ.	- Giảm thiểu sự dư thừa dữ liệu đảm bảo tính nhất quán. - Nhưng lại nhân bản dữ liệu đến các địa điểm mà các ứng dụng cần đến, giúp cho việc thực thi các ứng dụng không dừng nếu có một địa điểm bị hỏng. Từ đó vấn đề quản lý nhất quán dữ liệu sẽ phức tạp hơn.
Các cấu trúc vật lý phức tạp và truy xuất hiệu quả	Các cấu trúc vật lý phức tạp giúp cho việc truy xuất dữ liệu được hiệu quả.	Các cấu trúc vật lý phức tạp giúp liên lạc dữ liệu trong cơ sở dữ liệu phân tán .
Tính toàn vẹn, phục hồi, đồng thời	Dựa vào giao tác.	Dựa vào giao tác phân tán.

▼ DDB: Trong database tập trung không bao giờ dư thừa dữ liệu (do đã được chuẩn hóa csdl) còn bên CSDLPT chắc chắn sẽ vi phạm (Khi ta nhân bản nhưng trên db tập trung vẫn phải đạt chuẩn 3)

▼ DDB: Về mức độ an toàn và tin cậy thì CSDLPT tốt hơn CSDL vì ta chỉ đẩy những thông tin liên quan đến chi nhánh đó tránh việc lộ thông tin của chi nhánh khác.

▼ DDB: Các thành phần cần thiết cho việc xây dựng CSDLPT là:

1. Thành phần quản trị CSDL (DB Database Management)
2. Thành phần truyền dữ liệu (DC Data Communication)

3. Từ điển dữ liệu (DD Data Dictionary) mở rộng để biểu diễn thông tin về sự phân tán dữ liệu trên mạng
4. Thành phần CSDLPT (DDB) (nằm trong dịch vụ Agent)

▼ DB: Nếu khi nào CSDL tập trung không giải quyết được nhu cầu của Khách hàng thì mới cần CSDLPT

- Tốn kém chi phí server
- Lập trình dễ phát sinh lỗi, phức tạp hơn so với lập trình trên 1 CSDL
- Có khả năng xảy ra sự cố cho dù lập trình đúng.

▼ DDB: Tại sao phải phân mảnh?

Do nhu cầu thực tế của bài toán.

▼ DDB: Nhân bản (1 phần, toàn phần) → Khả năng hỗ trợ truy vấn rất cao → Cập nhật dữ liệu lâu

▼ DDB: Phân hoạch → Độ sẵn sàng thấp → Cập nhật nhanh.

▼ DDB: Các hình thức phân mảnh

- Phân mảnh ngang
- Phân mảnh dọc
- Hỗn hợp (ngang và dọc)

▼ DDB: Các quy tắc phân mảnh: trong quá trình phân mảnh phải tuân thủ 3 quy tắc để đảm bảo CSDL sẽ không thay đổi về mặt ngữ nghĩa (dữ liệu và ràng buộc toàn vẹn không bị mất)

Xét về quan hệ R chứ không xét trên góc độ DB

1. Tính đầy đủ
2. Tính tái thiết được
3. Tính tách biệt

→ Thông thường nếu 1 quan hệ R thỏa Tính đầy đủ thì cũng thỏa Tính tái thiết được.

▼ DDB: Phân mảnh ngang: Nguyên thủy và dẫn xuất

- Nguyên thủy: Là sự phân mảnh 1 quan hệ dựa trên 1 vị từ (điều kiện) được định nghĩa trên 1 quan hệ.
- Dẫn xuất: Là 1 quan hệ dựa vào các vị từ được định nghĩa trên 1 quan hệ khác.

▼ DDB: Phân mảnh dọc

Chia theo các cột. Để có khả năng tái thiết lại quan hệ nguyên thủy thì mỗi phân mảnh dọc phải chứa khóa chính của quan hệ nguyên thủy đó.

Trên thực tế phân mảnh dọc sẽ vi phạm tính tách biệt.

▼ DDB: Trong suốt phân tán

Nghĩa là 1 SP khi ta thực thi ở 1 Server phân mảnh bất kỳ thì vẫn thực thi được ở tất cả các Server phân mảnh còn lại mà ta không cần chỉ lại đường dẫn đến SP cần truy xuất (người dùng không cảm nhận được SP đang chạy trên Hệ thống phân tán → tưởng đang chạy trên hệ thống tập trung). ⇒ Nhân bản SP ở nơi cần thiết

▼ DDB: Ở mức độ người dùng thì họ phải đạt được mức trong suốt cao nhất → Trong suốt phân tán (Mức 1). Ở mức độ lập trình viên thì đứng ở mức 3 (Sự trong suốt ánh xạ cục bộ)

▼ DDB: Ta có thể đẩy những article nào sang các site phân mảnh?

- Table
- View
- SP
- UDF

▼ DDB: Để viết SP đạt sự phân tán thì ta cần nắm định nghĩa và 2 quy tắc:

- Định Nghĩa:
    - Nghĩa là 1 SP khi ta thực thi ở 1 Server phân mảnh bất kỳ thì vẫn thực thi được ở tất cả các Server phân mảnh còn lại mà ta không cần chỉ lại đường dẫn đến SP cần truy xuất (người dùng không cảm nhận

được SP đang chạy trên Hệ thống phân tán → tưởng đang chạy trên hệ thống tập trung). ⇒ Nhân bản SP ở nơi cần thiết

- Quy tắc:
  - Tên các CSDL ở Server Subcirber phải giống nhau.
  - Tên các LINKSERVER phải giống nhau.

▼ DDB: Khi Set 1 giá trị cho 1 biến thì nó chỉ nhận đúng 1 giá trị. Nếu như truyền nhiều giá trị vào thì nó sẽ lấy ngẫu nhiên 1 giá trị trong tập giá trị đó.

▼ DDB: Dùng Cursor để thao tác trên từng dòng từng dòng của Cursor thay vì 1 table như Bảng tạm. Cho phép gởi OUTPUT từ 1 SP cho 1 SP khác tiếp nhận và thực thi tiếp (thứ mà SP bình thường không làm được)

▼ DDB: Sau khi phân tán xong thì có 1 trường dữ liệu là rowguild. Vậy rowguid được sinh ra để làm gì?

> Đáp: Hỗ trợ quá trình đồng bộ dữ liệu từ site phân mảnh về site chủ và ngược lại.

▼ DDB: Login Name là gì? Username là gì?

> Login Name là tài khoản chúng ta dùng để đăng nhập vào một server. Ví dụ như tài khoản SA - tài khoản phổ biến khi đăng nhập SQL Server để tăng độ bảo mật. Sau khi kết nối thành công tới một server, chọn Security → Logins sẽ thấy tài khoản SA

> Username là tài khoản mà chúng ta dùng để làm việc trên một cơ sở dữ liệu. Ví dụ trên cơ sở dữ liệu QLVT, chọn vào QLVT → Security → Users sẽ thấy các tài khoản có thể làm việc trên cơ sở dữ liệu này

▼ DDB: db_dataReader, db_dataWriter, db_securityAdmin,.... rồi db_accessAdmin là gì ? Chúng dùng để làm gì ?

> Đáp: chúng là những quyền mà một tài khoản kiểu login name có thể được chỉ định & quyết định xem chúng có thể làm những gì trên cơ sở dữ liệu đó.

> 1.db_dataReader là quyền chỉ xem dữ liệu, không thể thêm mới hoặc sửa đổi dữ liệu

> 2.db_dataWriter là quyền chỉ ghi dữ liệu, dĩ nhiên nếu ghi được thì coi như cũng đọc được dữ liệu

> 3.db_securityAdmin là quyền tạo các tài khoản login name, tài khoản này dùng để đăng nhập vào server

> 4.db_accessAdmin là quyền tạo các tài khoản username, tài khoản dùng để thao tác trên một cơ sơ dữ liệu nhất định

> 5.db_owner là quyền cao nhất với một cơ sở dữ liệu, quyền này cho phép xem, thêm, xóa, sửa tất cả dữ liệu & có thể tạo tài khoản login name và username mà không có giới hạn nào(kiểu bạn là chủ nhà thì làm gì cũng được ấy)

▼ DDB: Nên viết SP ở phân mảnh hay ở Site gốc rồi đẩy về?

ĐÁP: Nên viết ở Site gốc rồi đẩy về. Vì khi ta muốn sửa 1 SP thì ta chỉ cần sửa ở Site chủ thì nó sẽ tự đồng bộ xuống các Site phân mảnh liên quan.

▼ DDB: Nếu sửa Stored Procedure trên phân mảnh thì nó có đồng bộ về các phân mảnh khác và server gốc, điều này là đúng hay sai ?

Đáp: Sai. Code chỉ đồng bộ theo một chiều từ server gốc tới các phân mảnh. Không có chiều ngược lại.

Chỉ có data là đồng bộ 2 chiều mà thôi.

▼ DDB: Muốn thực thi một câu lệnh store procedure, view, function thì ta làm như nào ?

> Đáp: Ta sẽ tạo một đối tượng là SqlCommand rồi nhúng vào các đối số cần thiết vào

▼ DDB: Giao tác là gì? Để viết giao tác cần bật dịch vụ gì?

> Đáp: Giao tác là một dãy các thao tác đọc và ghi dữ liệu cùng với các bước tính toán nhất quán (Begin Trans, Commit , Rollback,Begin distributed trans) để giải quyết các tình huống khi dữ liệu bị mất tính nhất quán khi có nhiều truy xuất đồng thời

> Để viết giao tác cần phải bật dịch vụ MDTC - Miscrosoft Distribute Transaction Coordinator

▼ DDB: Ý nghĩa của Begin trans, commit, rollback & Begin distributed trans là gì ?

> Đáp:

> Begin Trans: bắt đầu giao tác

> Commit : xác nhận thành công

> Rollback(hoặc abort) : hủy bỏ giao tác và trả lại dữ liệu cũ

> Begin distributed trans : câu lệnh mở đầu giao tác phân tán

▼ DDB: Nêu 4 tính chất giao tác ?

> Đáp: Có tất cả 4 tính chất với giao tác gồm:

> (1) tính nguyên tử: 1 giao tác có nhiều tập lệnh thì mọi câu lệnh hoặc thành công hoặc nếu 1 lệnh thất bại thì tất cả thất bại

> (2) tính nhất quán: đảm bảo tất cả rằng buộc. Mọi lệnh đều được chạy

> (3) tính biệt lập: một giao tác đang thực thi thì ko làm lộ các kết quả của nó cho những giao thức khác đang cùng hoạt động trước khi nó hoàn thành.

> (4) tính bền vững: một giao tác đã được commit thì dữ liệu đó được giữ nguyên và không thể rollback

▼ DDB: Các quy tắc phân mảnh: Đảm bảo 3 tính chất sau:

- Tính đầy đủ
- Tính tái thiết
- Tính tách biệt

▼ DDB: Dữ liệu rác là gì?

Đáp: là dữ liệu sinh ra có thể chưa phải là dữ liệu đã được commit nếu một giao tác khác đồng thời vào và lấy giao tác này ra thì dữ liệu này là ko chính xác.

▼ DDB: Có mấy loại giao tác?

> Đáp: Có 2 loại giao tác là tập trung và phân tán

> Giao tác tập trung có 2 loại: giao tác phẳng và giao tác lồng

> (1) Giao tác phẳng có điểm khởi đầu duy nhất (begin transaction) và một điểm kết thúc duy nhất (end transaction)

> (2) Giao tác lồng chứa nhiều giao tác với điểm khởi đầu và duy nhất riêng biệt.

> Giao tác phân tán chỉ có 1 loại duy nhất: giao tác phẳng

▼ DDB: XACT_ABORT là gì ? Nó nhận được mấy giá trị ?

> Đáp: XACT_ABORT: đây là tùy chọn kết nối. Nó chỉ nhận 2 giá trị là ON & OFF.

> Nếu là OFF, SQL Server sẽ bỏ qua lệnh gây lỗi trong transaction và tiếp tục thực thi các lệnh còn lại.

> Nếu là ON, SQL Server sẽ hủy bỏ toàn bộ transaction nếu nó gặp lệnh bị lỗi và trả lại dữ liệu về ban đầu.

▼ DDB: Khi viết 1 stored procedure khi nào ta không dùng begin transaction , không dùng commit ... nhưng vẫn được coi là 1 giao tác ?

Đáp: khi ta sử dụng chỉ một lệnh UPDATE - DELETE - INSERT duy nhất. Còn nếu có 2 lệnh UPDATE - DELETE - INSERT thì phải dùng cú pháp begin trans và commit

▼ DDB: Vị từ thích hợp là gì ?

> Đáp: Thỏa tính đầy đủ và tính cực tiểu

▼ DDB: Tiêu chí đầy đủ và tiêu chí cực tiểu là gì ?

> Đáp:
>
> Tiêu chí đầy đủ là ta có nhiều stored procedure thì xác suất nó truy xuất tới các phân mảnh là như nhau.
>
> Tiêu chí cực tiểu là mỗi site được sinh ra phải được sử dụng tới trong một stored procedure nào đó.

▼ DDB: Vị từ là gì ? Một vị từ đơn giản là gì ?

> Đáp: Vị từ là Mệnh đề logic có nhiều điều kiện
>
> Một vị từ đơn giản là vị từ có kiểu: MACN = 'CN1',... . Tức vế trái là tên thuộc tính và vế phải là tên giá trị.
>
> Một vị từ sơ cấp: MACN = (CN1 hội CN2) Hội của các vị từ đơn giản.

▼ DDB: Một vị từ "thích hợp" là gì ?

Đáp: Một vị từ "thích hợp" nghĩa là khi tạo ra 1 phân mảnh "thích hợp" thì phải có stored procedure sử dụng tới phân mảnh này.

▼ DDB: Sự trong suốt phân tán là gì ?

> Đáp: Sự trong suốt phân tán: khi 1 stored procedure(sp) được thực thi ở 1 server phân mảnh bấy kì thì cũng hoạt động tốt trên các phân mảnh còn lại - người dùng không cảm thấy được sp đang chạy trên hệ thông phân tán.
>
> Note: Chúng ta đứng ở vị trí thứ 3 để lập trình. Luôn ưu tiên tìm ở phân mảnh cục bộ trước khi sang các phân

> mảnh khác.

▼ **DDB: Có mấy mức độ trong suốt phân tán là gì ?**

> Đáp: Có 4 mức độ trong suốt phân tán

> (1) Mức cao nhất: Không cần chỉ ra phân mảnh cần truy vấn. Người dùng không cảm giác là đang thao tác trên một câu truy vấn phân tán.

> (2) Sự trong suốt vị trí: Chúng ta chỉ cần chỉ ra tên TABLE cần phải truy vấn.

> (3) Sự trong suốt ánh xạ cục bộ: Chúng ta cần chỉ rõ tên phân mảnh và vị trí cấp phát của chúng

> (4) Không trong suốt: Chúng ta phải viết lệnh để đến đúng vị trí database cần thao tác. Tuy nhiên, do chúng ta đã đặt tên các cơ sở dữ liệu giống nhau ở mọi phân mảnh. Tên LINK server cũng trùng tên nên không phải bận tâm đến mức này

> Note: người dùng sẽ sử dụng ứng dụng ở mức cao nhất. Còn người lập trình như chúng ta sẽ đứng ở mức thứ 3.

▼ **DDB: Điều kiện để có thể phân tán được cơ sở dữ liệu? Có mấy bước để phân tán cơ sở dữ liệu?**

> Đáp: có dịch vụ SQL Server Agent

> Có 3 bước để thực hiện phân tán cơ sở dữ liệu

> (1) Định nghĩa Server Distributor : chứa Database distribution

> (2) Định nghĩa publication : 1 container chứa các article (table, view, stored procedure, UDF).

> Trong đó, phải chỉ rõ các server :

> Publisher - chứa bản gốc cơ sở dữ liệu

> Distributor - điều phối phân tán dữ liệu

> (3) Định nghĩa subscription (database): 1 container nhận publication. Trong đó, phải chỉ rõ server Subscriber - chứa cơ sở dữ liệu sau khi phân tán

▼ **DDB: Nêu đặc điểm của phân mảnh ngang ?**

> Đáp: Phân mảnh ngang là chia một quan hệ theo các bộ. Mỗi phân mảnh ngang là tập con của quan hệ.

> Trong phân mảnh ngang bao gồm phân mảnh ngang nguyên thủy & phân mảnh ngang dẫn xuất

> Phân mảnh ngang nguyên thủy là phân mảnh của 1 quan hệ dựa trên 1 vị từ được định nghĩa trên quan hệ đó

> Phân mảnh ngang dẫn xuất là phân mảnh của 1 quan hệ dựa trên vị từ được định nghĩa trên quan hệ khác.

> Ví dụ với đề quản lý vật tư thì

> MaChiNhanh = 'CN1' → phân mảnh ngang nguyên thủy

> NHANVIEN.MaKho = KHOHANG.MaKho → phân mảnh ngang dẫn xuất

> Nói cách khác 'CN1' và 'CN2' là các vị từ.

▼ **DDB: Nêu đặc điểm của phân mảnh dọc ?**

> Đáp: Phân mảnh dọc là phân mảnh dựa trên khóa chính của một quan hệ (phải có khóa chính để đảm bảo tính tái thiết)

> Giả sử, NHANVIEN(id, ho , ten, dia chi, luong, ngay sinh, ma chi nhanh) thì khi tạo phân mảnh dọc sẽ thành

> NHANVIEN(id, ho, ten, ngay sinh, ma chi nhanh) trong đó MaChiNhanh là khóa ngoại thì phải cho phép NULL

▼ **DDB: Nêu đặc điểm của phân mảnh hỗn hợp ?**

Là sự kết hợp của 2 cách phân mảnh ngang và dọc. Khi hợp các phân mảnh theo 2 cách này thì cơ sở dữ liệu tạo thành chính là cơ sở dữ liệu ban đầu.

▼ **DDB: Có mấy quy tắc phân mảnh?**

> Đáp: Có 3 quy tắc phân mảnh

> (1) Tính đầy đủ: xét trên góc độ là QUAN HỆ. Mỗi mục dữ liệu ít nhất phải nằm ở 1 phân mảnh. Thì không bị mất thông tin

> (2) Tính tái thiết: thường thì thỏa mản tính đầy đủ sẽ thỏa mãn tính tái thiết.

> (3) Tính tách biệt: mỗi mục dữ liệu chỉ nằm ở một phân mảnh duy nhất. Khi ghép các phân mảnh thì database sẽ

> đầy đủ như ban đầu. Tức nếu TEN ở phân mảnh 1 thì sẽ không xuất hiện ở phân mảnh 2

> Note Thực tế, tính tách biệt dễ bị vi phạm vì NHÂN BẢN giúp dữ liệu tồn tại ở cả 2 mảnh. Bắt buộc tính đầy đủ phải thỏa mãn. Nếu thỏa mãn thì tính tái thiết cũng sẽ được bảo đảm.

▼ **DDB: Snapshot folder là gì ?**

Đáp: Folder chứa dữ liệu trung gian để đồng bộ dữ liệu từ các phân mảnh về site chủ và ngược lại. Là folder chứa dữ liệu để đẩy qua đẩy lại. Phải là network path(shared folder)

▼ **DDB: Run on continue khác run on demand ở điểm nào ?**

Đáp: Run on continue làm tính nhất quán cao, dữ liệu đồng bộ ngay lập tức. Run on demand tính tự quản cao. Các site có thể disconnect. Các thay đổi không phản ảnh tức thời tới site chủ

▼ **DDB: Có những cách nào để tối ưu hóa truy vấn ?**

> Đáp: Có 5 cách để tối ưu hóa một câu truy vấn

> Dùng phép chọn, chiếu trước, phép kết sau

> Khử phép kết (nếu được)

> Nếu 1 điều kiện xuất hiện nhiều lần trong WHERE thì dùng các phép biến đổi tương đương để cho điều kiện đó xuất hiện 1 lần

> Trong mệnh đề AND, điều kiện nào có xác suất sai cao thì đặt ở đầu ; OR thì ngược lại.

> Field tham gia trong điều kiện truy vấn nên được sắp thứ tự trước & thứ tự này phải đc sử dụng trong mệnh đề truy vấn với WITH (INDEX=ten_index)

▼ DDB: Thế nào là một Stored Procedure trong suốt ? Điều kiện để viết Stored Procedure trong suốt?

> Đáp: Stored Procedure được coi là trong suốt nếu ta cho thực thi ở 1 server thì vẫn cho thực thi được ở những server còn lại mà không cần chỉ rõ đường dẫn đến table cần truy xuất

> Hoặc có thể trả lời

> Stored Procedure trong suốt là Stored Procedure mà khi ta cho thực thi ở bất kì phân mảnh nào đều cho kết quả giống nhau

> Điều kiện để viết Stored Procedure trong suốt là ta cần có LINK SERVER & tên database giống nhau

▼ DDB: Trong database, cái nào là nhân bản, cái nào là phân hoạch ?

> Đáp: Mở cây dẫn xuất ra xem, cái nào không có trong cây dẫn xuất là nhân bản, cái đầu tiên là phân mảnh ngang nguyên thủy

▼ DDB: Nếu đã phân tán xong cơ sở dữ liệu, muốn thay đổi cấu trúc | thứ tự các cột trong table của server gốc thì làm sao?

Có thể thay đổi bằng cách viết stored procedure dùng lệnh ALTER TABLE ở server gốc sau đó đồng bộ xuống các server phân mảnh

▼ DDB: Trong các table, cái nào mang tính đầy đủ, cái nào vi phạm tính tách biệt?

Đáp: Các table nhân bản thì vi phạm tính tách biệt, tất cả các table còn lại thì mang tính đầy đủ.

▼ DDB: Giao tác tập trung với giao tác phân tán giống và khác nhau như thế nào?

> Đáp:

> Điểm khác : Giao tác thì thực thi trên môi trường cơ sở dữ liệu tập trung (gồm có giao tác phẳng và giao tác lồng), còn giao tác phân tán thì thực thi trên môi trường cơ sở dữ liệu phân tán.

> Điểm giống : 4 tính chất giao tác.

▼ DDB: Login Name nằm trong table nào?

Đáp: Nằm trong table sys.syslogins trong Database đó

▼ DDB: Tại sao biết user liên kết với login nào?

Đáp: Username và loginname liên kết với nhau qua trường sid (trên user và login đều có sid) nên từ loginname biết được username từ sid

▼ DDB: Tên nhóm quyền nằm trong table nào ?

> Đáp: Nằm trong table sys.sysuser trong Database đó

▼ DDB: Ưu | khuyết điểm của nhân bản ?

> Đáp: Truy xuất nhanh, đứng ở đâu cũng có thể select được. Nhưng update chậm vì có quá nhiều bản sao

▼ DDB: Ưu | khuyết điểm của phân hoạch ?

> Đáp: Select chậm nhưng insert và update nhanh do chỉ thao thác trên server gốc hoặc server phân mảnh cần truy xuất.

▼ DDB: Tính sẵn sàng của nhân bản cao hơn so với phân hoạch

▼ DDB: Sự bố tải khi truy cập trên nhiều site phân mảnh:

Khi cần truy xuất 1 dữ liệu.

Nếu ta truy cập vào site chủ tìm chắc chắn sẽ có nhưng thời gian truy xuất lâu vì quá nhiều dữ liệu.

Nhưng nếu truy xuất trên nhiều site phân mảnh cùng lúc sẽ nhanh hơn rất nhiều vì chúng có nhiều cpu để xử lý đồng thời và dữ liệu của mỗi site là rất bé.

▼ DDB: Mục đích thiết kế phân mảnh là xác định các phân mảnh không chồng chéo lên nhau.

▼ DDB: Đã tạo ra 1 phân mảnh thì phải có ít nhất 1 ứng dụng chạy trên nó.

▼ DDB: Nhập liệu (gồm simple (không có khóa ngoại, có khóa ngoại), subform, fill), Báo cáo (Thống kê), Quản trị (gồm login, register)

▼ DDB: Tối ưu hóa truy vấn trên hệ thống phân tán là cho đồng thời nhiều site chạy cùng 1 lúc ⇒ Giảm bớt thời gian chạy rất nhiều so với chạy tuần tự trên hệ thống tập trùng.

▼ Tối ưu hóa: AND nếu cái nào xác suất sai cao hơn thì để trước, OR thì cái nào xác suất đúng cao hơn thì để trước.

▼ DDB: Tối ưu hóa

1. Dùng phép chọn/chiếu trước, phép kết sau
2. Khử phép kết (nếu được): Nếu kết quả in ra 1 cột mà dữ liệu không thay đổi thì bảng cung cấp dữ liệu cho một cột có thể truy vấn riêng.
3. Nếu 1 đk xuất hiện nhiều lần trong WHERE thì dùng các phép biến đổi tương đương để cho đk đó xuất hiện 1 lần.

   P1 ^ (P2 vP3) ≡ (P1 ^ P2) v (P1 ^ P3)

P1 v(P2^P3) ≡ (P1 vP2)^ (P1 vP3)

P1 ^ (P1 v P2 v P3) ≡ P1

P1 v (P1 ^ P2 ^ P3) ≡ P1

(1) $P_1 \wedge P_2 \equiv P_2 \wedge P_1$
(2) $P_1 \vee P_2 \equiv P_2 \vee P_1$
(3) $P_1 \wedge (P_2 \wedge P_3) \equiv (P_1 \wedge P_2) \wedge P_3$
(4) $P_1 \vee(P_2 \vee P_3) \equiv (P_1 \vee P_2) \vee P_3$
(5) $P_1 \wedge (P_2 \vee P_3) \equiv (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$
(6) $P_1 \vee(P_2 \wedge P_3) \equiv (P_1 \vee P_2) \wedge (P_1 \vee P_3)$
(7) $\neg(P_1 \wedge P_2) \equiv \neg P_1 \vee \neg P_2$
(8) $\neg(P_1 \vee P_2) \equiv \neg P_1 \wedge \neg P_2$
(9) $\neg(\neg P) \equiv P$

(10) $P \wedge P \equiv P$
(11) $P \vee P \equiv P$
(12) $P \wedge true \equiv P$
(13) $P \vee false \equiv P$
(14) $P \wedge false \equiv false$
(15) $P \vee true \equiv true$
(16) $P \wedge \neg P \equiv false$
(17) $P \vee \neg P \equiv true$
(18) $P_1 \wedge (P_1 \vee P_2 \vee P_3) \equiv P_1$
(19) $P_1 \vee (P_1 \wedge P_2 \wedge P_3) \equiv P_1$

Vídụ: Xét truy vấn Q10 ở trên, điều kiện q ở dạng chuẩn hợp là:

$(\neg P_1 \wedge P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2 \wedge \neg P_2) \vee P_3$

Bằng cách áp dụng phép biến đổi (16), chúng ta được:

$(false \wedge \neg P_2) \vee (\neg P_1 \wedge false) \vee P_3$

Khi các biểu thức con đã được xác định, chúng ta có sử dụng các phép biến đổi tương đương sau đây để đơn giản hóa một cây toán tử:

(1) $R \bowtie R \equiv R$
(2) $R \cup R \equiv R$
(3) $R - R \equiv \varnothing$
(4) $R \bowtie \sigma_F(R) \equiv \sigma_F R$
(5) $R \cup \sigma_F(R) \equiv R$
(6) $R - \sigma_F(R) \equiv \sigma_{\neg F}(R)$
(7) $\sigma_{F1}(R) \bowtie \sigma_{F2}(R) \equiv \sigma_{F1 \wedge F2}(R)$
(8) $\sigma_{F1}(R) \cup \sigma_{F2}(R) \equiv \sigma_{F1 \vee F2}(R)$
(9) $\sigma_{F1}(R) - \sigma_{F2}(R) \equiv \sigma_{F1 \wedge \neg F2}(R)$
(10) $R \cap R \equiv R$
(11) $R \cap \sigma_F(R) \equiv \sigma_F R$
(12) $\sigma_{F1}(R) \cap \sigma_{F2}(R) \equiv \sigma_{F1 \wedge F2}(R)$
(13) $\sigma_F(R) - R \equiv \varnothing$

Ý nghĩa của các phép biến đổi này là loại bỏ các phép toán dư thừa.

4. Trong mệnh đề AND, điều kiện nào có xác suất sai cao hơn thì đặt ở đầu, mệnh đề OR điều kiện nào có xác suất đúng hơn thì đặt ở đầu.
5. Field tham gia điều kiện truy vấn nên được sắp thứ tự trước, và thứ tự này phải được sử dụng với WITH.
6. Hạn chế Distinct, Order By: Hạn chế sử dụng 2 từ khóa này. Hai phép oán này chiếm phần lớn thời gian truy vấn vì phải thực hiện lọc bản ghi trùng và sắp xếp các bản ghi. Dùng INDEX thay ORDER BY.
7. Tránh dùng func trong toán tử so sánh.
8. Cân nhắc việc loại bỏ các câu truy vấn con tương quan (≠ truy vấn lồng).
9. Sử dụng FORCESEEK trong các truy vấn dùng toán tử LIKE hoặc IN.

▼ DDB: Muốn chạy 2 SP cùng lúc thì chạy SP1 sau đó cho SP2 chạy trong 1 job.

▼ DDB: Tạo Jobs (AGENT) có 2 mục đích: 1 là chạy song song, 2 là định nghĩa 1 lệnh khi đúng ngày đó nó sẽ chạy tự động (backup nên cho tự động chạy).

▼ AGENT có 3 mục đích:

- Phân tán CSDL

- Hỗ trợ đồng bộ csdl trên môi trường phân tán
- Hỗ trợ cho Jobs chạy.

▼ (?): Nếu SP trả số liệu về thì làm sao nhận số liệu đó để tổng hợp?

▼ (?): Nếu SP chạy song song với Job, nếu cả 2 cùng trả về kết quả thì xử lý như thế nào, làm thế nào để cả 2 chạy xong rồi mới làm tổng hợp vì có trường hợp cả 2 chưa chắc đã chạy xong cùng lúc?

▼ DDB: 3 phần của giao tác

1. Khái niệm giao tác, định nghĩa hình thức của giao tác

   Giao tác được xem như một dãy các thao tác đọc **và ghi** trên cơ sở dữ liệu cùng với các bước tính toán cần thiết (Begin Trans, Commit, Rollback, Begin Distributed Trans) để đảm bảo tập lệnh như 1 đơn vị lệnh.

2. Các tính chất của giao tác:
   a. Tính nguyên tử (atomicity):
      i. Xem 1 SP thực thi 1 công việc gồm nhiều lệnh nhưng xem như là 1 đơn vị lệnh. (All-or-Nothing). Để thực hiện được tính nguyên tử thì phải bật **XACT_ABORT.**

   b. Tính nhất quán (consistency):
      i. Tính đúng đắn của SP khi thực thi, đảm bảo các ràng buộc toàn vẹn trước và sau khi chạy giao tác được đảm bảo.
      ii. Khi chạy tất cả những lệnh thành công không có trường hợp là dừng giữa chừng.

   c. Tính biệt lập (isolation):
      i. Khi nó đang hoạt động không cho phép các thiết bị khác thấy được kết quả đang thực thi.

   d. Tính bền vững (durability):

i. Sau khi Commit (ủy thác) thì không thể rollback được. Nếu có sự cố xảy ra vẫn không thể hoàn tác. Còn nếu mà đang chạy giữa chừng bị cúp điện thì SP đó coi như chưa hoạt động (dữ liệu được rollback).

Hay chúng còn được gọi là ACID.

3. Phân loại giao tác:
   a. Giao tác tập trung:
      i. Giao tác phẳng: SP đó chỉ có 1 giao tác, nếu roll back là rollback hết, commit là commit hết
      ii. Giao tác lồng: trong giao tác có giao tác
   b. Giao tác phân tán
      i. Giao tác phẳng.

▼ DDB: (?) Khi người ta cho mình 1 bài toán, làm sao biết nó là giao tác tập trung phẳng hay lồng?

Chúng ta dùng giao tác lồng khi sẽ có 1 phần được commit 1 phần được rollback. Còn rollback hết commit hết thì dùng phẳng.

▼ DDB: Khi nào không cần cài giao tác (Begin trans) nhưng giao tác vẫn hoạt động? ⇒ Sp đó chỉ có 1 lệnh cập nhật duy nhất, insert hoặc update hoặc delete.

▼ DDB: Khi nào cần cài giao tác cho SP? ⇒ Khi sp có từ 2 lệnh cập nhật trở lên.

▼ DDB: Thế nào là dữ liệu rác?

Giả sử 1 máy đang chạy SP thay đổi giá trị của 1 field trên 1 table thì cùng thời điểm 1 máy khác muốn lấy dữ liệu của table đó với field đang được thay đổi là dữ liệu mới. Một lúc sau vì 1 lý do nào đó SP rollback trả về dữ liệu củ thì dữ liệu mới mà máy thứ 2 đang sử dụng gọi là dirty read (dữ liệu rác).

▼ DDB: XACT_ABORT là gì?

Đây là tùy chọn ở mức kết nối, chỉ có tác dụng trong phạm vi kết nối của ta. XACT_ABORT nhận hai giá trị ON và OFF (OFF là giá trị mặc định). Khi tùy chọn này được đặt là OFF, SQL Server sẽ chỉ hủy bỏ lệnh gây ra lỗi trong

transaction và vẫn cho các lệnh khác thực hiện tiếp, nếu lỗi xảy ra được đánh giá là không nghiêm trọng. Còn khi XACT_ABORT được đặt thành ON, SQL Server mới cư xử đúng như mong đợi – khi gặp bất kỳ lỗi nào nó hủy bỏ toàn bộ transaction và quay lui trở lại như lúc ban đầu.

▼ DDB: Giao tác phân tán chỉ có 1 giao tác phẳng và nếu như hầu hết site đều chạy thành công nhưng 1 site thất bại thì nó phải roll back trên tất cả các site.

▼ DDB: Rollback là sẽ trở về trạng thái cũ hay là sẽ chạy ngược lại quá trình? ⇒ Không cập nhật trực tiếp trên table gốc mà làm trên vùng nhớ tạm, khi commit thì nó sẽ cập nhật dữ liệu trên vùng nhớ tạm vào table cơ sở.

▼ DDB: (?) Khi viết giao tác tập trung thì rollback khi chạy sai nhưng trên giao tác phân tán không có lệnh rollback vậy thì lệnh rollback đó hoạt động như thế nào? đặt ở đâu? ai điều khiển?

⇒ Distributed Transaction Coordinator sẽ làm việc đó.

▼ DDB: 1 trang dữ liệu là 4KB (4096 Bytes)

▼ DDB: Article chứa UDF, TABLE, SP, VIEW

▼ DDB: BindingSource là gì?

- Là nơi chỉ ra (Binding) để thao tác dữ liệu, là cầu nối giữa DataGridView và DataTable

▼ DDB: DataSet, DataTable, DataAdapter là gì?

- DataSet có thể hiểu là một bản ghi tạm, nó lưu trữ và chỉnh sửa dữ liệu ở Local Cache, tức là khi có ngắt kết nối tới server đi nữa thì vẫn thao tác với dữ liệu trên DataSet. Sau khi xem sét và chính sửa dữ liệu xong ta tạo kết nối (DataAdapter) để Update dữ liệu từ Local(DataSet) về DataSource(SQL)
- Dữ liệu trong DataSet được lưu ở dạng DataTable(DT). DT là một đối tượng chứa dữ lệu từ DataBase gởi qua, cấu trúc gồm có các DataRow và DataColumn. DT có thể là 1 Table, 1 View, hoặc 1 SP, chứa dữ lệu nhưng không chứa lệnh.

- DataAdapter(DA) là một dạng cầu nối, giúp các lệnh như Select, Insert, Update, Delete để trao đổi dữ liệu giữa Client và Server, và DA đổ dữ liệu từ DataSource về DS bằng phương thức Fill, GetData

▼ DDB: Thuộc tính folder REPLDATA là gì? Ở đâu? Mục đích của thư mục dùng để làm gì?

Thuộc tính là share. Có quyền read/write. Mục đích chứa các dữ liệu trao đổi trong quá trình update data từ site gốc về các site phân mảnh và ngược lại.

▼ DDB: Isolation Level (Mức biệt lập)

5 mức biệt lập:

1. Read Uncommited
2. **Read Commited**
3. Repeatable Read (chỉ cản được lệnh update chứ không cản được INSERT hay DELETE).
4. Serializable
5. Snapshot (như Serializable chỉ khác cách xử lý)

Mức Isolation	Dirty read	Nonrepeatable read	Phantom read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable	No	No	No
Snapshot	No	No	No

▼ DDB: Chẳng lẽ lúc nào viết giao tác cũng phải chèn thêm đoạn Isolation Level vô khối lệnh? Ta có thể quy định database ở mức số 4 hoặc 5 thông qua câu lệnh:

```
ALTER DATABASE QLVT_DATHANG
SET ALLOW_SNAPSHOT_ISOLATION ON
```

▼ DDB: Các lệnh EndEdit, ResetCurrentItem sử dụng trong bài có ý nghĩa gì?

- AddNew() thêm một item/row vào danh sách,
- CancelEdit() bỏ qua các dữ liệu đang chỉnh sửa trong rrơ và tả con trỏ về row cuối cùng. Chỉ có tác dụng với lệnh AddNew và dữ liệu đang được hiệu chỉnh nhưng chưa ghi vào DataSet.
- EndEdit: kết thúc quá trình chỉnh sửa dữ liệu và gơi và DataSet.
- ResetCurrentItem: Reread/ đọc lại row hiện tại ta đang đứng, dữ liệu hiện tại ở row ta đang đuungs không còn ở dạng tạm nữa(tức là dữ liệu có thể cancle edit) mà hiện thị dữ liệu đã chính thức ghi vào DataSet
- Update cập nhật data từ DataSet về DB thông qua TableAdapter.
- Count(): đếm xem số record tương ứng với row đang chọn tồn tại bao nhiêu bảng trong bindingsource được kiểm tra.
- RemoveCurrent Xóa row đang chọn ra khỏi DataSet

▼ Khi nào cần phải tạo link server? Khi tạo cần các ràng buộc gì?

- Tạo link server khi database nằm ở 2 server khác nhau , ràng buộc là: tên các server link tới nhau thì phải cùng tên.
- Mục đích: cho phép truy cập dữ liệu từ server này sang server khác
- Cách tạo: ví dụ tạo link ở server 1 để truy xuất dữ liệu ở server 2
- Đứng ở server 2 tạo login HTKN, đứng ở server 1 tạo LINK
- Số link : n(n-1) ; n là số server phân mảnh, số tên link : n-1
- Cú pháp: TENLINK.TENDATABASE.DBO.TENBANGTRUYXUAT

▼ DDB: user trong server và login

- Trong 1 server có nhiều user – tên user có thể trùng nhau

- Trong 1 database có nhiều user – tên user ko đc trùng nhau
- Trong 1 server có nhiều login- tên login ko đc trùng nhau
- 1 login chỉ thuộc 1 user

▼ DDB: Các mức cô lập dữ liệu

- Read Uncommitted - mức yếu nhất - TT1 đang sửa đổi & TT2 có thể vào xem dữ liệu ngay cả khi TT1 chưa lưu thay đổi lại. Nói nôm na là "tôi không cần biết dữ liệu có đang được cập nhật hay không, hãy cho tôi dữ liệu hiện có ngay tại thời điểm này".
- Read Commited - mức mặc định - TT1 đang sửa đổi dữ liệu thì TT2 không thể xem dữ liệu này. Chỉ khi TT1 xong thì TT2 mới xem được dữ liệu. Nếu TT2 sửa dữ liệu thì TT1 sẽ có được dữ liệu được sửa đổi từ TT2.
- Repeatable Read - mức an toàn - TT2 đang đọc một bảng dữ liệu thì TT1 không thể chỉnh sửa bảng dữ liệu này cho đến khi TT2 hoàn tất việc đọc dữ liệu đó. Nói nôm na là dữ liệu đang được đọc sẽ được bảo vệ khỏi cập nhật bởi các transaction khác
- Serializable - mức an toàn cao - TT1 đang sửa đổi một bảng dữ liệu thì TT2 không thể làm bất cứ hành động gì với bảng dữ liệu đó như: INSERT, UPDATE, DELETE,.....
- Snapshot - mức an toàn cao nhất - hoạt động tương tự Serializable, TT1 vẫn hoạt động, TT2 sẽ sửa dữ liệu trên một bản ghi đã được Snapshot sao chép.

▼ Đồ án

▼ Tại sao table đó (VatTu) nhân bản?

Trong đề vật tư thì vật tư nhân bản vì vật tư có thể có ở chi nhánh này cũng có thể có ở chi nhánh khác.

▼ Tại sao phải tạo Data Source?

- Để fill dữ liệu từ CSDL vào các table bên trong → Không cần tạo class rồi đẩy dữ liệu vào List xong mới đẩy lên view.

▼ Tại sao khi phân tán nên để Vật tư, DDH, PN, PX không nên theo NV mà theo KHO? Vì NHÂNVIÊN có thể bị điều chuyển từ

CHINHANH này sang CHINHANH khác → DDH, PN, PX tự động di chuyển theo.

▼ Khi truy vấn dữ liệu, nếu trên site đang đứng mà không tìm thấy thì nên về site chủ

▼ Khi không tìm được trên server hiện tại, em muốn sang phân mảnh khác để tìm thay về server gốc có được không ?

> Đáp: Có 2 lý do để không nên sử dụng cách này

> Lý do 1: Về lý thuyết thì được nhưng nếu chẳng may server phân mảnh đó offline thì sẽ không hoạt động được. Ngược lại, server gốc thì luôn online.

> Lý do 2: Giả sử có 3 phân mảnh và thông tin cần tìm ở server 3. Tuy nhiên, khi viết stored procedure thì chúng ta lại tìm kiếm theo trình tự 1 → 2 → 3. Nếu chẳng may server 2 không hoạt động thì câu lệnh sẽ trả về lỗi và chấm dứt thực thi luôn. Trong khi server 3 thì lại hoạt động bình trường. Do đó về server gốc là hợp lý hơn.

▼ Chi nhánh với user đăng nhập vào thì thấy hết tất cả đơn đặt hàng , phiếu nhập, phiếu xuất luôn hả thầy, hay chỉ mỗi phiếu của chính họ thôi hay sao?

> Đáp: Thấy hết, nhưng chỉ cho chỉnh sửa các phiếu do mình lập

▼ DDB: Trên 1 quan hệ nhân bản giả sử có Mã vật tư và muốn tìm số lượng tồn của mã vật tư đó thì tìm trên site hiện tại nếu không có thì tìm ở các site tiếp theo nếu không có thì báo lỗi còn nếu có thì in ra số lượng tồn là đúng hay sai?

SAI. Khi đã nhân bản thì ở mọi nơi đều có như nhau nên nếu site hiện tại đã không có thì các site khác cũng không có.

▼ DDB: Nếu 1 người thuộc chức vụ cao nhất (Công ty) thì có quyền xem dữ liệu của tất cả chi nhánh vậy có cần tạo tk cho người đó ở mỗi site hay không?

KHÔNG. Nếu tạo ở 1 site sau đó tự động thêm tài khoản đó ở các site còn lại sẽ gặp vấn đề nếu như tk đó đã tồn tại ở 1 site khác → Không thể tạo được.

Vì thế ta chỉ cần tạo tk ở 1 site sau đó thông qua tk HOTROKETNOI để truy cập dữ liệu của các site còn lại.

▼ DDB: Tại sao trả về mã nhân viên khi dùng sp_DangNhap ?

> Đáp: để gán tự động cho các form có mã nhân viên

▼ DDB: Bốn thuộc tính quan trọng của combo box?

> Đáp:

> data source chứa danh sách dữ liệu cung cấp cho nó

> display member chứa tên field mình muốn hiển thị

> value member chứa field dữ liệu mình muốn lấy

> selected index change phương thức thay đổi giá trị

▼ DDB: Vì sao ta phải dùng Remote Login ?

> Đáp: Cho phép truy cập dữ liệu khi đứng từ server này sang server khác.

▼ DDB: Sắp xếp nhân viên theo tên, họ trong view rp → tạo index cho Ten, Họ asc. Không nên dùng ORDER BY (chắc chắn không phải QUICK SORT)

▼ DB: Những phần mềm ứng dụng trên CSDL thường có 3 phần (Nhập liệu, báo cáo, quản trị (login, sign up, backup)
▼ Project Tại sao lại Sign up trên sql server mà không tạo 1 table tài khoản như thông thường người ta hay làm? ⇒ Về mặt CSDL nếu tạo ra table username riêng để quản lý thì đang tìm ẩn 1 loại lỗi, với lỗi này người đăng nhập vào phần mềm nếu họ hiểu cơ cấu truy vấn của App thì họ sẽ gửi 1 chuỗi kí tự sẽ gây vô hiệu hóa pwd. Đăng nhập mà chẳng cần pwd.