ChatGPT

Khóa học Oracle 12c DBA qua đối thoại Dev & DBA

Ngữ cảnh: Một lập trình viên (Dev) mong muốn chuyển sang vai trò Quản trị Cơ sở dữ liệu Oracle 12c (DBA). Nội dung được trình bày dưới dạng cuộc trò chuyện giữa Dev và một DBA giàu kinh nghiệm, đi từ kiến trúc cơ bản đến các chủ đề nâng cao, với những lời khuyên thực tế và một chút "chửi nhẹ" vui vẻ nhằm nhấn mạnh bài học.

Giới thiệu và Mở đầu

Dev: (Háo hức) Em chào anh! Em là lập trình viên nhưng đang muốn chuyển hướng sang làm DBA Oracle 12c. Nghe nói phải học nhiều lắm, chắc tầm vài tuần?

DBA: (Cười) Chào em. Ừ, hành trình trở thành DBA không ngắn đâu – khóa học này dự kiến khoảng **100 giờ** đấy!

Dev: 100 giờ lận ạ?

DBA: Đúng rồi, làm DBA không dễ đâu em. Chúng ta phải đi từ những khái niệm nền tảng nhất đến các kỹ thuật nâng cao. Nhưng đừng lo, anh sẽ hướng dẫn từng bước, kiểu trò chuyện thoải mái. Em cứ hình dung như nghe podcast giữa hai anh em mình vậy.

Dev: Ngon quá, dạng trao đổi thế này dễ tiếp thu hơn hẳn. Em sẵn sàng rồi anh ơi!

DBA: Tốt. Mục tiêu của khóa này là giúp em *tư duy như một DBA*, hiểu rõ Oracle 12c từ gốc đến ngọn: từ **kiến trúc, cài đặt, kết nối, vận hành**, đến **quản lý schema, quyền hạn, tablespace**, rồi **tối ưu truy vấn, sao lưu & phục hồi, điều chỉnh hiệu năng**. Đồng thời anh sẽ chỉ ra sự khác biệt giữa cách nghĩ của Dev và DBA, cũng như những lỗi "kinh điển" mà dev hay mắc phải trong mắt DBA. Sẵn sàng chưa?

Dev: Dạ sẵn sàng! Bắt đầu thôi anh.

Kiến trúc Oracle 12c: Từ đơn nhân đến đa tenant

Dev: Trước giờ em làm web với MySQL, thường mỗi ứng dụng một database đơn giản. Oracle 12c nghe nói kiến trúc khác hẳn, có cả "container" gì đó? Anh giải thích giúp em với.

DBA: Đúng rồi. Oracle 12c giới thiệu một **kiến trúc hoàn toàn mới gọi là Multitenant (đa tenant)** – thay đổi lớn nhất kể từ Oracle 8i luôn [1]. Thay vì mỗi cơ sở dữ liệu (CSDL) là một thực thể độc lập hoàn toàn với **một instance** riêng (instance = bộ nhớ SGA + các process nền), từ 12c ta có thể tạo **CSDL dạng "đa tenant"**. Nôm na là một **Container Database (CDB)** chứa nhiều **Pluggable Database (PDB)** bên trong.

Dev: Container với Pluggable? Nó khác gì so với cách cũ anh?

DBA: Để anh so sánh: Ở các phiên bản cũ (như 11g trở về trước), nếu em muốn chạy 2 CSDL trên một máy thì phải khởi tạo 2 instance riêng, mỗi cái có đủ bộ nhớ SGA, tiến trình nền (DBWR, LGWR, v.v.). Tài nguyên bị chia nhỏ ra và nhiều thành phần trùng lặp. 12c giải quyết bằng cách cho **nhiều "CSDL con" (PDB)** chạy trong cùng một **CSDL cha (CDB)** dùng chung instance. Hình dung như thay vì chạy 5 cái ô tô tốn 5 động cơ, giờ gom thành 5 toa trên một đoàn tàu xài chung một đầu máy – **chia sẻ tài nguyên** và quản lý tập trung [2] [3].

Dev: À, ví dụ hay đó anh – nhiều người đi chơi thay vì đi 5 xe thì gộp đi 1 xe 16 chỗ cho tiết kiệm .

DBA: Chính xác. Với kiến trúc mới:

- **CDB** có các thành phần chung: SGA (bộ nhớ), background processes, datafile chung cho **metadata của container** (gọi là dictionary chung).
- Mỗi **PDB** chứa dữ liệu ứng dụng riêng, có datafiles riêng, nhưng **dùng chung** tài nguyên CDB.

Nghĩa là thay vì 5 CSDL độc lập tranh giành tài nguyên, giờ chỉ 1 instance "kéo" cả 5 PDB, giảm bớt trùng lặp (ví dụ chỉ có một DBWR, một LGWR dùng chung) [4]. Việc **quản trị** cũng nhẹ nhàng hơn: vá lỗi, nâng cấp chỉ làm trên CDB là các PDB hưởng hết, khỏi lặp lại 5 lần [5].

Dev: Hay thật. Vậy 12c có 2 chế độ: non-CDB (kiểu cũ) và CDB (đa tenant)?

DBA: Đúng. Khi cài đặt Oracle 12c em có thể chọn tạo *database kiểu cũ (non-CDB)* hoặc *dạng CDB có PDB*. Thực tế Oracle định hướng multitenant là tương lai.

Dev: Nếu nhiều PDB chung 1 CDB, lỡ một PDB die thì các cái khác có sao không anh?

DBA: Mỗi PDB logic thì tách biệt dữ liệu, user,... nhưng dùng chung instance nên nếu instance die (ví dụ máy chủ mất điện) thì **toàn bộ CDB và các PDB đều ngừng**. Còn nếu chỉ một PDB bị lỗi dữ liệu thì các PDB khác không ảnh hưởng vì dữ liệu tách riêng. Em có thể backup, restore từng PDB độc lập nữa.

Dev: Ra vậy. Ngoài vụ multitenant, kiến trúc Oracle chắc còn nhiều thứ khác? Anh nói qua giúp em hệ thống nó gồm những phần gì với.

DBA: Oke. Kiến trúc Oracle gồm 2 thành phần chính: **Instance** và **Database**.

- **Instance**: là **phần chạy trong memory (RAM)**, gồm **SGA (System Global Area)** và các **background process**. SGA chứa các vùng nhớ dùng chung (như buffer cache để cache dữ liệu, shared pool cache câu lệnh SQL, v.v.), các process nền thì bao gồm: DBWR (ghi buffer cache xuống đĩa), LGWR (ghi nhật ký giao dịch – redo log), CKPT (đánh dấu checkpoint), SMON, PMON (quản lý session, recovery...), v.v. Tất cả những thứ này kết hợp gọi là *instance*. Nói nôm na, instance là **cầu nối giữa người dùng và dữ liệu vật lý trên đĩa** [2].

- **Database (CSDL)**: là **tập hợp các tệp tin vật lý trên đĩa**: bao gồm **data files** (chứa dữ liệu bảng, index), **control file** (lưu thông tin cấu trúc CSDL, checkpoint, v.v.), **redo log files** (nhật ký giao dịch để khôi phục khi cần). Oracle lưu mọi thứ quan trọng vào các file này. Khi instance hoạt động, nó *mở* các file đó lên.

Dev: Vậy instance + data files = một database hoạt động.

DBA: Đúng. Đó là khái niệm cơ bản: Instance (SGA+process) **mount** database (mở các file). Khi em lệnh `startup`, Oracle sẽ khởi tạo instance (nomount), sau đó *mount* controlfile (mount), rồi *open* datafiles (open).

Dev: Mấy trạng thái nomount, mount, open này dev bọn em ít nghe thật.

DBA: Ừ, nhưng DBA thì hay dùng khi khởi động hoặc phục hồi. Em chỉ cần nhớ: **Oracle instance có thể chạy mà chưa mở CSDL (nomount)** – thường để thực hiện tạo controlfile, phục hồi, v.v. Còn bình thường phải mở CSDL (open) thì user mới truy cập data được.

Dev: Rõ. Thế SGA, PGA là gì anh?

DBA: SGA là vùng nhớ dùng chung cho mọi process – ví dụ khi một câu SQL truy cập dữ liệu, Oracle sẽ đọc data block từ đĩa vào **buffer cache (trong SGA)**, lần sau query khác cần dữ liệu đó thì lấy luôn từ SGA, khỏi đọc đĩa nữa cho nhanh. SGA còn chứa **shared SQL areas** – chỗ chứa các câu lệnh SQL đã parse, để reuse nếu query lặp lại [6].

Dev: Còn **PGA**?

DBA: PGA (Program Global Area) là vùng nhớ riêng cho mỗi **process server** (tiến trình phục vụ user). Oracle có hai loại process: **user process** (chạy trên client, ví dụ app của em) và **server process** (chạy trên máy DB, đại diện để xử lý request của user process). Mỗi server process có PGA riêng (chứa thông tin session, vùng sort dữ liệu cục bộ, v.v.), không chia sẻ giữa các process.

Dev: Em hiểu rồi: SGA = "bếp chung", PGA = "bếp riêng" cho từng thằng .

DBA: Ha, ví von hay đấy. Đúng là **SGA giống một gian bếp chung** chứa các nguyên liệu, công cụ phổ biến để mọi đầu bếp (process) dùng cho tiện, còn PGA là quầy riêng mỗi đầu bếp làm việc của mình [7].

Dev: Vậy Oracle quản lý SGA/PGA ra sao?

DBA: Em có thể cấu hình kích thước SGA, PGA. Oracle 12c có chế độ **tự động quản lý bộ nhớ** (Automatic Memory Management) – chỉ cần đặt tổng dung lượng, Oracle tự chia cho SGA/PGA và điều chỉnh các thành phần bên trong SGA (buffer cache, shared pool…) tối ưu. Lúc cài thường ta chọn luôn option này cho khỏe.

Dev: Good. Em nghĩ tạm vậy là ổn phần kiến trúc chung. Anh cho em xem hình tổng quan kiến trúc Oracle 12c được không?

DBA: Có đây. Hình dưới minh họa các **thành phần kiến trúc Oracle 12c**, bao gồm SGA, PGA, các process nền và cách chúng tương tác với các file dữ liệu:

Sơ đồ kiến trúc Oracle Database 12c (đơn instance, non-CDB) với các thành phần chính: cấu trúc bộ nhớ (SGA, PGA), các tiến trình nền (DBWn, LGWR, SMON, PMON,…), và các tệp dữ liệu vật lý (data files, redo logs, control file).

Dev: (Xem hình) Oái, chi tiết phết anh ha – đủ thứ process.

DBA: Ừ, nhiều thật. Em không cần nhớ hết ngay đâu. Quan trọng là hiểu **Oracle phức tạp hơn các hệ quản trị CSDL nhẹ như MySQL** vì nó **giàu tính năng và tối ưu hiệu năng cho hệ thống lớn**. DBA phải nắm được các thành phần này để chẩn đoán và cấu hình về sau.

Dev: Dạ. Còn trong kiến trúc 12c, phần multitenant (CDB/PDB) có hình không anh?

DBA: Có, đây:

Kiến trúc multitenant của Oracle 12c: Một Container DB (CDB) có chung instance và các tablespace hệ thống (CDB$ROOT, chứa dữ liệu hệ thống), chứa nhiều Pluggable DB (PDB) bên trong. Mỗi PDB là một CSDL con, có dữ liệu ứng dụng riêng (tablespace ứng dụng), dùng chung tài nguyên với CDB.

Dev: (Gật gù) Nhìn hình dễ hiểu hơn nhiều, kiểu một ông lớn chứa nhiều ông nhỏ bên trong.

DBA: Chuẩn. Em để ý trong CDB có **PDB$SEED** – đó là PDB mẫu, Oracle dùng nó làm khuôn để tạo PDB mới (tạo rất nhanh).

Dev: À, như template sẵn.

DBA: Đúng. Thôi, vậy tạm ổn phần kiến trúc. Chốt lại: Oracle 12c có cải tiến **đa tenant** giúp quản lý nhiều CSDL dễ dàng hơn [8]. Em cần hiểu instance vs database, SGA/PGA, CDB/PDB. Giờ ta sang phần cài đặt & kết nối nhé.

Hướng dẫn cài đặt Oracle 12c và cách kết nối cơ bản

Dev: Anh ơi, cài Oracle chắc phức tạp hơn cài MySQL nhỉ? Em cài MySQL phát là chạy, còn nhớ lần trước thử cài Oracle mà rối quá, nào là Oracle Home, Listener loạn cả.

DBA: (Cười) Đúng là Oracle "nặng đô" hơn. Để anh hướng dẫn:

1. Chuẩn bị cài đặt:
- Xác định hệ điều hành: Oracle chạy tốt nhất trên **Linux/Unix**. Bản cho Windows cũng có nhưng ít dùng production. Dev thì có thể cài trên Windows cho tiện học, hoặc dùng máy ảo Linux.
- Cấu hình máy: Oracle 12c yêu cầu RAM kha khá (tối thiểu ~2GB để chạy ổn), đĩa vài chục GB trống.

2. Cài đặt phần mềm Oracle:
- Tải bộ cài Oracle Database 12c (ví dụ 12.1 hoặc 12.2) từ trang Oracle. Lưu ý phải có tài khoản Oracle.
- Giải nén và chạy trình cài đặt (Oracle Universal Installer). Trong quá trình cài, em sẽ chọn **ORACLE_BASE** (thư mục gốc chứa mọi thứ) và **ORACLE_HOME** (thư mục chứa phần mềm Oracle).
- Chọn tạo **Database mới** luôn trong quá trình cài (có DBCA – Database Configuration Assistant hỗ trợ). Ở bước này em có thể chọn tạo CDB với PDB hay non-CDB.

Dev: Lúc trước em thấy có bước đặt **SID** nữa?

DBA: Đúng. **SID (System Identifier)** là tên định danh instance. Em đặt tên SID (ví dụ ORCL). Nếu chạy nhiều Oracle instance trên một máy, mỗi cái phải có SID khác nhau.

Dev: Vậy sau khi cài xong, Oracle sẽ chạy luôn ạ?

DBA: Thường lúc cài xong, nó tạo xong database và mở instance luôn. Em sẽ có một **database mẫu** (ví dụ ORCL) với các user mặc định (SYS, SYSTEM...).

Dev: Người mới như em nên cài dạng desktop hay server anh?

DBA: Em có thể cài **Oracle Database 12c Standard/Enterprise** trên máy cá nhân để học. Hoặc dùng bản **Oracle 12c Express Edition (XE)** gọn nhẹ (nếu có 12c XE, hình như 11g với 18c có XE). Cách cài tương tự, XE thì đơn giản hơn.

Dev: Cảm ơn anh. Phần cài đặt em sẽ làm theo hướng dẫn. Giờ giả sử CSDL đã chạy, làm sao **kết nối** vào Oracle hả anh?

DBA: Đây là chỗ nhiều dev mới gặp bỡ ngỡ nè. Oracle có cơ chế kết nối riêng qua **Oracle Net** (giao thức TNS). Để ứng dụng hoặc client kết nối DB, cần có **Listener** và chuỗi kết nối đúng định dạng:

- **Oracle Listener:** Là một tiến trình chạy trên server, lắng nghe trên cổng (mặc định 1521). Nó giống "nhân viên trực cổng", khi client yêu cầu kết nối tới DB thì listener nhận và điều phối **server process** phục vụ client đó [9] [10].

- **Chuỗi kết nối (connect string):** Thay vì chỉ host:port/db như MySQL, Oracle dùng chuỗi dạng `USER/PASS@Host:Port/ServiceName` hoặc sử dụng **TNS name** được cấu hình trong file `tnsnames.ora`.

Dev: Em nghe về `tnsnames.ora` rồi – nó làm gì anh?

DBA: File `tnsnames.ora` chứa **danh sách các kết nối (TNS alias)** tới các DB. Mỗi alias có các thông tin: **tên service (hoặc SID)** của DB, **hostname**, **port**... [11]. Ví dụ có đoạn:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = dbserver)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = ORCL))
  )
```

Alias `ORCL` này định nghĩa kết nối tới DB service ORCL trên máy `dbserver` cổng 1521. Khi đó từ client, em chỉ cần gõ `sqlplus user/pass@ORCL` là nó đọc `tnsnames.ora` để biết kết nối tới đâu.

Dev: Nghĩa là Oracle cho phép đặt alias dễ nhớ thay vì gõ nguyên chuỗi dài.

DBA: Đúng. Dev hay nhầm chỗ này: **trong Oracle "database" != "schema"** (mình sẽ nói kỹ sau), và **ServiceName != SID** trong nhiều trường hợp. Với CDB/PDB, ServiceName thường là tên PDB để kết nối vào đúng PDB.

Dev: Cụ thể hơn được không anh?

DBA: Ví dụ: CDB tên *ORCLCDB* chứa PDB tên *ORCLPDB*. Trên server có listener. Em muốn kết nối **vào PDB** thì ServiceName phải là ORCLPDB (vì PDB đăng ký service đó với listener). SID của instance có thể là ORCLCDB nhưng **đừng** dùng SID đó nếu muốn vào PDB (trừ khi kết nối dưới dạng SYSDBA).

Dev: Vâng. Nói chung chắc bắt buộc phải có listener thì client mới vào được.

DBA: Đúng. Mà thường cài Oracle xong người ta cấu hình listener luôn. Có thể có nhiều listener trên một máy nếu nhiều DB, nhưng thường 1 listener xài chung cũng được [12].

Dev: Em dùng **Oracle SQL Developer** để kết nối chắc dễ hơn ha – chỉ cần nhập host, port, SID.

DBA: Ừ, công cụ GUI đó giúp quản lý kết nối dễ dàng, nó cũng tạo sẵn file tnsnames khi mình lưu kết nối. Với người dùng cuối hoặc dev thì xài SQL Developer được, nhưng **DBA cũng nên biết xài dòng lệnh** qua tool `sqlplus`.

Dev: À `sqlplus` huyền thoại.

DBA: Chuẩn. Ví dụ, để kết nối bằng SQL*Plus tới DB cục bộ, em có thể:
- Mở cmd/terminal, export ORACLE_SID=ORCL (trên Linux) rồi gõ: `sqlplus / as sysdba` – kết nối local bằng OS authentication với quyền SYSDBA (dành cho DBA).
- Hoặc kết nối remote: `sqlplus user/password@//dbserver:1521/ORCL` (với ORCL là ServiceName).

Dev: Hiểu rồi. Vậy chắc làm quen dần.

DBA: Một điểm quan trọng: **Quyền SYSDBA** – khi em kết nối `as sysdba` tức là đăng nhập với quyền siêu admin, thường chỉ tài khoản `SYS` hoặc user được cấp quyền SYSDBA mới dùng được. Lúc cài Oracle nó hỏi mật khẩu cho **SYS và SYSTEM**, nhớ không?

Dev: Dạ nhớ. Em hay đặt đại, chắc toàn quyền admin hở anh?

DBA: (Nghiêm giọng) Tuyệt đối **không được tùy tiện dùng user SYS/System để làm việc hằng ngày**. SYS có quyền tối thượng, *DBA của DBA* luôn. **SYSTEM** cũng có role DBA nhưng **không có SYSDBA**, tức là không tự startup/shutdown instance được nếu không có SYSDBA [13]. Nói chung, để an toàn:
- **SYS**: chỉ dùng cho các tác vụ quan trọng (startup, shutdown, phục hồi, nâng cấp).
- **SYSTEM**: dùng cho quản trị chung, nhưng cũng tránh dùng để tạo bảng ứng dụng.

Dev: Vậy bọn em dev nếu muốn user riêng thì sao?

DBA: Thì tạo user ứng dụng riêng (ví dụ user `APP` nào đó) chứ ai lại dùng SYSTEM. Dùng user admin làm bừa rồi lỡ xóa nhầm gì trong schema hệ thống thì **toang**.

Dev: Dạ, tại nhiều dev không có DBA kèm nên cứ dùng luôn tài khoản admin cho tiện .

DBA: Hiểu. Môi trường dev thì có thể xài đại, nhưng lên production DBA sẽ cấp user riêng có quyền hạn hạn chế cho ứng dụng chạy. Cái này lát phần quản lý user & quyền anh sẽ nói kỹ.

Dev: Ok anh. Vậy tóm gọn phần kết nối: Oracle dùng **listener** và **chuỗi TNS**. Dev cần làm quen định dạng kết nối, có thể thông qua `tnsnames.ora`. Và nhớ đừng dùng user SYS/System lung tung.

DBA: Chuẩn rồi.

Vận hành cơ sở dữ liệu: Start/Stop và các tác vụ cơ bản

Dev: Giờ giả sử DB Oracle đã cài và chạy. Làm DBA chắc phải biết cách **bật/tắt CSDL (startup/shutdown)** và các công việc vận hành hàng ngày nhỉ?

DBA: Đúng thế. Đầu tiên là **quản lý instance**:

- Để **khởi động** CSDL Oracle, em đăng nhập `sqlplus / as sysdba` trên server rồi dùng lệnh `STARTUP`. Có thể chỉ `STARTUP` (mặc định mở DB luôn), hoặc `startup mount` / `nomount` tùy mục đích. Thông thường DBA chỉ cần `STARTUP` để đưa DB lên trạng thái open cho user dùng.

- Để **dừng (shutdown)**, dùng lệnh `SHUTDOWN [IMMEDIATE]`. Tùy chọn **IMMEDIATE** rất hay dùng: nó kết thúc các session đang chạy một cách nhã nhặn (các transaction dang dở sẽ rollback) rồi đóng DB. Nếu dùng `SHUTDOWN ABORT` thì giống "rút điện đột ngột" – chỉ khi khẩn cấp vì nó sẽ cần recovery khi mở lại.

Dev: Mỗi lần tắt mở như vậy cần login bằng sysdba hở anh?

DBA: Đúng, chỉ user có quyền SYSDBA hoặc SYSOPER mới startup/shutdown được [14]. Ứng dụng bình thường không làm việc này.

Dev: Có khi nào DBA phải restart DB thường xuyên không?

DBA: Trong môi trường chuẩn thì Oracle server thường chạy **liên tục 24/7**, trừ khi bảo trì định kỳ hoặc gặp sự cố. DBA cố gắng hạn chế downtime.

Dev: Vậy nếu có lỗi, ví dụ DB bị treo, mình xử lý sao?

DBA: Oracle ít khi treo lắm, nếu có thường do vấn đề tài nguyên (hết RAM, full disk, v.v.). DBA phải **theo dõi hệ thống** để phòng tránh: ví dụ thấy **tablespace đầy** phải thêm dung lượng trước, thấy **RAM thấp** phải điều chỉnh SGA, v.v. Nếu xui treo, thì dùng `shutdown abort` rồi startup lại, sau đó kiểm tra **alert log** xem lỗi gì.

Dev: Alert log là gì anh?

DBA: Alert log là file log hệ thống của Oracle, ghi lại các sự kiện quan trọng (lỗi, thông báo, start/stop DB, lỗi ORA-xxxx nghiêm trọng...). Nó nằm trong thư mục `diag` của Oracle. Mỗi khi có lỗi hay hành động gì đáng chú ý, Oracle ghi vào đó. DBA **mỗi ngày nên kiểm tra alert log** để sớm phát hiện vấn đề.

Dev: Ồ, giống log của ứng dụng, nhưng đây là log của database.

DBA: Đúng. Ngoài ra còn log **listener** (listener.log) ghi các kết nối, và **trace files** ghi chi tiết lỗi cho từng process khi gặp sự cố.

Dev: Vận hành còn gì nữa anh?

DBA: Nhiều thứ: **quản lý không gian (tablespace)**, **theo dõi hiệu năng**, **backup**,... Nhưng những cái đó ta có mục riêng. Ở đây anh muốn nhấn mạnh quy trình làm việc giữa Dev và DBA một chút:

- Ở môi trường doanh nghiệp, thường có các môi trường **Dev/Test/Staging/Prod** riêng. DBA chịu trách nhiệm tạo và vận hành các môi trường DB đó.
- **Triển khai thay đổi**: Khi Dev cần thay đổi cấu trúc DB (ví dụ alter table, thêm chỉ mục) trên production, thường **DBA sẽ xem xét và thực hiện theo lịch**. Dev **không tự ý** làm trên prod. Đây là điểm khác biệt lớn trong tư duy: Dev thì muốn "đẩy nhanh cho xong yêu cầu", còn DBA thì ưu tiên an toàn và ổn định hệ thống.

Dev: Nghe quen à nha. Nhiều khi bọn em bảo "sao mấy anh DBA khó khăn, thêm cái index thôi mà bắt đợi cuối tuần" .

DBA: (Cười) Đó, dev các chú chỉ thấy mỗi user của chú, không thấy "bức tranh lớn". Thêm index lúc giờ cao điểm có thể khóa bảng, chặn cả trăm người dùng khác. Nên DBA mới hẹn lịch ban đêm hoặc cuối tuần. Em nhớ câu này: **"Yes you can hack the schema nhanh cho kịp deadline, *nhưng* làm bừa vậy sẽ phá hỏng những gì chúng tôi đang xây dựng cho dữ liệu dài hạn"** [15]. DBA luôn nghĩ đến **tác động lâu dài và toàn cục**, không chỉ một tính năng nhỏ trước mắt.

Dev: Vâng, góc nhìn khác nhau thật.

DBA: Vì thế công việc vận hành còn là **cầu nối với team dev**:
- DBA hỗ trợ tạo các **bản sao DB test** (clone từ backup) để dev test cho sát thực tế.
- Rồi DBA duyệt các script thay đổi do dev viết, đảm bảo tuân thủ chuẩn và **có kế hoạch backout** (phương án lùi nếu có sự cố) [16].

Dev: Em hình dung ra quy trình release có DBA tham gia để mọi thứ trơn tru, an toàn.

DBA: Chuẩn. Một khía cạnh khác là **bảo mật & phân quyền** trong vận hành nữa, nhưng để phần sau nói. Giờ em đã nắm:
- Cách start/stop cơ bản,
- Luôn xem alert log khi nghi vấn,
- Cẩn trọng với thay đổi trên production (phối hợp dev-DBA).

Chúng ta chuyển sang phần quản lý **schema và storage** nhé – rất quan trọng đối với dev lên DBA.

Quản lý Schema và Tablespace trong Oracle

Dev: Hồi nãy anh có nhắc *"schema trong Oracle không giống database"*. Em vẫn hơi lẫn: Schema là gì?

DBA: Hiểu đơn giản: **Schema = User** trong Oracle. Mỗi user Oracle đồng thời là một schema (tập hợp các bảng, index, thủ tục, v.v. do user đó sở hữu).

Dev: Tức là nếu em **tạo một user** mới, Oracle sẽ cấp cho user đó một *schema rỗng* cùng tên để chứa đối tượng?

DBA: Đúng rồi. Ví dụ em `CREATE USER dev1 ...;` rồi `GRANT CREATE TABLE TO dev1;`. Khi dev1 kết nối và tạo bảng `dev1.mytable`, bảng đó thuộc schema dev1.

Trong Oracle, **một database (CDB hoặc non-CDB)** có thể chứa nhiều user/schema độc lập. Các schema này có thể **phân quyền** cho nhau để truy cập nếu muốn, còn mặc định là tách biệt.

Dev: Khác hẳn MySQL - bên MySQL mỗi database kiểu như một schema riêng luôn.

DBA: Ừ. Oracle mạnh hơn ở chỗ em có thể host nhiều ứng dụng trong cùng một database bằng các schema khác nhau. Tuy nhiên thời nay với multitenant, có xu hướng mỗi ứng dụng một PDB. Dù sao khái niệm schema/user vẫn vậy.

Dev: Vậy **quản lý schema** gồm những gì anh?

DBA: Gồm **tạo user, cấp quota, phân quyền** để các schema hoạt động đúng nhu cầu.

- Khi tạo một user (schema) mới, DBA sẽ **chỉ định**:
- **Tablespace mặc định** để lưu các đối tượng của user đó (vd: `DATA_TBS`).
- **Temporary tablespace** để phục vụ các phép sort, join tạm thời.

Ví dụ lệnh:

```
CREATE USER dev1 IDENTIFIED BY "pwd123"
  DEFAULT TABLESPACE USERS
  TEMPORARY TABLESPACE TEMP
  QUOTA 100M ON USERS;
```

Lệnh này tạo user *dev1*, mật khẩu *pwd123*, mặc định dùng tablespace USERS, temp TS là TEMP, và giới hạn dung lượng 100MB trên TS USERS.

Dev: À, có **QUOTA** nữa – để làm gì anh?

DBA: Quota giới hạn dung lượng user được phép dùng trong một tablespace nhất định. Như ví dụ trên, dev1 chỉ được dùng 100MB ở tablespace USERS. Nếu vượt quá, Oracle chặn không cho tạo thêm dữ liệu (báo lỗi ORA-01536).

Dev: Hiểu rồi, để tránh user xài hết không gian đĩa.

DBA: Đúng. Việc **gán tablespace và quota** là rất quan trọng. Oracle quản lý lưu trữ thông qua **tablespace**.

Dev: Giờ anh giải thích rõ **tablespace** giúp em.

DBA: Được. **Tablespace** là một **đơn vị lưu trữ lôgic** trong Oracle, gồm một hoặc nhiều data file ở mức vật lý [17]. Mọi dữ liệu (bảng, index, v.v.) đều được lưu trong một tablespace nào đó.

Em có thể hình dung:
- Tablespace giống như **ngăn tủ lớn**.
- Bên trong ngăn tủ đó (tablespace) là các **file dữ liệu** chứa đồ.

Khi tạo bảng, em có thể chỉ định nó nằm trong TS nào. Oracle sẽ lấy không gian trong datafile thuộc TS đó để lưu dữ liệu.

Dev: Mặc định khi tạo user thì hay cho nó vào tablespace USERS hả anh?

DBA: Ừ, Oracle có sẵn tablespace **SYSTEM, SYSAUX** (cho dữ liệu hệ thống), **TEMP** (cho dữ liệu tạm), **UNDOTBS** (lưu Undo data cho transaction), và thường có tablespace **USERS** cho user thường dùng. DBA có thể tạo thêm TS khác tùy nhu cầu.

Quan trọng: **Không nên lưu dữ liệu ứng dụng vào SYSTEM tablespace**. Cái này dev hay mắc phải khi tạo user nhưng quên đặt default TS, Oracle sẽ mặc định SYSTEM – rất nguy hiểm vì có thể làm đầy tablespace hệ thống. Luôn đặt default TS riêng (như USERS hoặc tự tạo).

Dev: Rõ. Thế ngoài quản lý không gian, tablespace còn ý nghĩa gì khác?

DBA: Nó còn giúp **tối ưu hiệu năng và quản trị**:

- DBA có thể **phân tách dữ liệu**: ví dụ tạo tablespace riêng cho từng ứng dụng, hoặc tách index vào tablespace khác để quản lý I/O hiệu quả hơn.
- Có thể **di chuyển (transport)** tablespace giữa các DB khác nhau (miễn cùng platform) để sao chép dữ liệu nhanh.
- **Backup/Recovery** có thể theo tablespace (backup tablespace độc lập, hoặc restore 1 tablespace khi cần).

Dev: Oracle hay thật, linh hoạt hơn hẳn kiểu mọi thứ chung một file của mấy DB nhỏ.

DBA: (Gật đầu) Quản trị storage trong Oracle là cả một chủ đề sâu. Em cũng cần biết khái niệm **Bigfile vs Smallfile tablespace**:
- **Smallfile TS** (mặc định) cho phép nhiều file, mỗi file tối đa ~32GB (với block size 8KB).
- **Bigfile TS** chỉ 1 file nhưng file đó có thể cực lớn (hàng trăm GB tới TB).

Dev: Sao phải bigfile anh?

DBA: Bigfile dùng khi quản lý storage cấp phát qua hệ thống khác (ASM, hoặc thiết bị lưu trữ), muốn giảm số lượng file Oracle quản lý. Còn smallfile thì linh hoạt thêm file khi cần.

Dev: Em đoán dev bọn em ít động vào tablespace, nên dễ không để ý.

DBA: Đúng, đây là **"tư duy DBA"**: luôn quan tâm dữ liệu nằm ở đâu, dùng bao nhiêu, có nguy cơ đầy không. Dev thường chỉ lo code chạy, còn DB đầy đĩa hay không ít ai nghĩ .

Dev: Chuẩn anh... tới lúc truy vấn lỗi mới tá hỏa gọi DBA.

DBA: Chuẩn luôn. Nhiều dev không hiểu khi thấy lỗi **"tablespace full"** hoặc **"ORA-01536: quota exceeded"** liền kêu DB bị gì. Thực ra do họ **chèn quá nhiều dữ liệu vượt quota**. Cho nên khi làm DBA, em phải **quy hoạch đủ dung lượng** và cấp quota phù hợp cho user, đồng thời **theo dõi mức sử dụng** thường xuyên.

Dev: Dạ, phải tập suy nghĩ kiểu đó.

DBA: Quay lại quản lý schema:

Khi đã có user/schema, dev có thể tạo các đối tượng trong đó (table, index...). Nhiệm vụ DBA là hỗ trợ nếu họ cần thêm quyền (như quyền CREATE TABLE đã nói), hoặc tạo các cấu trúc lớn (ví dụ tạo tablespace partition, etc.), và giữ mọi thứ **có trật tự**.

Dev: "Trật tự" ý anh là sao?

DBA: Nghĩa là **đặt quy ước**: ví dụ đặt tên schema, đặt tên tablespace, không để dev muốn tạo gì thì tạo lung tung. Chẳng hạn:
- Ứng dụng A có schema `APP_A`, dùng tablespace `APP_A_DATA` và `APP_A_INDEX`.
- Ứng dụng B schema `APP_B`...

Như vậy quản lý dễ, nhìn tên biết của ai, khi backup hay di chuyển cũng gọn.

Dev: Hay. Cái này dev ít khi nghĩ tới, thường đặt tên lung tung lắm .

DBA: Đó, nên cần DBA định hướng.

Tổng kết phần này: Em phải nắm được:
- **Schema = User**, mỗi user có default tablespace, temp tablespace, quota.
- **Tablespace** là đơn vị lưu trữ logic gồm datafile vật lý, giúp quản lý không gian lưu trữ.
- DBA phải thường xuyên kiểm tra dung lượng tablespace (xem có autoextend không, datafile đầy chưa) để mở rộng kịp lúc.

Dev: Dạ.

Quản lý Người dùng và Phân quyền (User & Privileges)

Dev: Anh nói thêm về quản lý user và quyền đi. Hôm trước em có đọc tài liệu, Oracle có khái niệm **System Privilege, Object Privilege, Role** phức tạp hơn MySQL nhiều.

DBA: Đúng rồi. Oracle rất mạnh về bảo mật phân quyền.

- **Hệ quyền (Privilege)** trong Oracle chia thành hai loại: **quyền hệ thống (system privileges)** và **quyền đối tượng (object privileges)** [18].
- *Quyền hệ thống* cho phép thực hiện các hành động ở cấp DB, ví dụ: `CREATE USER` (tạo user khác), `ALTER DATABASE`, `CREATE TABLE` (tạo table trong schema của mình), `SELECT ANY TABLE` (select trên mọi table trong DB), v.v. Có khoảng >100 quyền hệ thống.
- *Quyền đối tượng* là quyền trên các đối tượng cụ thể: ví dụ quyền `SELECT, UPDATE, DELETE` trên một bảng cụ thể, hoặc `EXECUTE` trên một stored procedure, v.v.
- **Role (vai trò):** Để quản lý dễ hơn, Oracle cho phép tạo **role** – tập hợp các quyền. Mình gán role cho user, thay vì gán lắt nhắt từng quyền. Oracle có sẵn một số role mặc định (như CONNECT, RESOURCE – ở bản cũ, 12c trở đi deprecated mấy role này). DBA thường tự tạo role theo nhu cầu. Ví dụ tạo role `DEV_ROLE` gồm quyền create session, create table, select any dictionary,... rồi grant role đó cho các user dev [19].

Dev: Nghe chuyên nghiệp ghê. MySQL tụi em chỉ có user với GRANT đơn giản.

DBA: Khác biệt môi trường enterprise đó em. Oracle còn có **PROFILE** để giới hạn tài nguyên theo user (như giới hạn số session, thời gian CPU, v.v.), đặt chính sách mật khẩu (mật khẩu mạnh, hết hạn,...). Nhưng cái đó nâng cao, anh chỉ mention để em biết.

Dev: Vâng. Thế khi tạo user Oracle mới, tối thiểu phải làm gì anh?

DBA: Tối thiểu:
1. `CREATE USER ... IDENTIFIED BY ... DEFAULT TABLESPACE ... TEMP ... QUOTA ...` – tạo user.
2. `GRANT CREATE SESSION TO user` – quyền cơ bản cho phép login.
3. Nếu user đó cần tạo object (bảng, procedure) thì cấp các quyền: `GRANT CREATE TABLE, CREATE PROCEDURE,... TO user`.
4. Nếu cần truy cập đối tượng của user khác: dùng `GRANT` trên từng đối tượng hoặc role.

Dev: Ví dụ user ứng dụng chỉ cần connect và SELECT/INSERT/... các bảng của chính nó thì sao?

DBA: Nếu user sở hữu bảng thì mặc định nó có full quyền trên bảng của nó rồi, không cần grant. Chỉ khi user muốn truy cập bảng thuộc schema khác, schema kia phải `GRANT SELECT ON tableX TO user`.

Dev: Em nhớ rồi.

DBA: Em cũng nên biết về các **tài khoản đặc biệt**:
- **SYS**: siêu tài khoản, sở hữu toàn bộ dictionary. Có mọi quyền, mặc định có role DBA và quyền SYSDBA.
- **SYSTEM**: có role DBA (hầu hết các quyền hệ thống) nhưng không có SYSDBA. Dùng cho quản trị chung.
- **SYSMAN**: tài khoản quản trị Oracle Enterprise Manager.
- **OUTLN**: dùng cho tính năng Outline cũ.
- ...

Khi nhận một hệ thống, DBA thường **kiểm tra các user mặc định** (Oracle có nhiều user built-in), khóa hoặc xóa những user không dùng để bảo mật (ví dụ **HR, SCOTT** – schemas mẫu).

Dev: Vụ bảo mật này chắc quan trọng lắm.

DBA: Rất quan trọng. Quan điểm DBA là **nguyên tắc "least privilege"** – cấp ít quyền nhất có thể để user/ứng dụng vẫn chạy được.

Ví dụ: ứng dụng web chỉ cần xài user A có quyền SELECT/INSERT/UPDATE trên các bảng của nó. Không lý do gì cấp cho nó quyền CREATE TABLE hay (tệ hơn) role DBA cả! Vậy mà anh từng thấy dev setup ứng dụng kiểu: connect bằng user SYSTEM cho "tiện" => **toang**.

Dev: (Ngại ngùng) Dạ em thú thật hồi mới làm có dùng SYSTEM thiệt, do lười tạo user...

DBA: (Lắc đầu) Đó là **tội đồ lớn nhất trong mắt DBA** đấy . Dùng account quyền quá cao cho app chẳng khác nào mở toang cửa cho hacker nếu app bị lộ thông tin kết nối. Em thử nghĩ, injection mà chạy bằng user DBA thì coi như mất cả database.

Dev: Rút kinh nghiệm sâu sắc luôn.

DBA: Good. Một số **sai lầm phổ biến của dev về quyền hạn**:

- *Dùng tài khoản DBA cho ứng dụng*: như nói ở trên, cực kỳ nguy hiểm.
- *Grant quyền lung tung*: ví dụ `GRANT ALL ON table TO user` – nhìn có vẻ tiện nhưng **ALL** bao gồm cả DROP, ALTER table... Không nên chút nào. Thay vào đó, chỉ cấp những gì cần (SELECT/INSERT...).
- *Không phân biệt môi trường*: Dev hay lấy tài khoản và pass giống nhau cho dev/test/prod, thậm chí share nhau dùng. DBA sẽ bắt phải tách biệt và đổi mật khẩu định kỳ.

Dev: Chuẩn, dev thì hay "tiện" cho nhanh, chứ DBA phải kỹ.

DBA: Em nói đúng, dev thiên về nhanh, DBA thiên về chắc chắn.

Dev: À anh, Oracle còn có **SYSDBA, SYSOPER** – hai cái đó như vai trò đặc biệt hả?

DBA: Đó là **quyền đặc biệt**, không phải role thường. Quyền **SYSDBA** cho phép làm *mọi thứ*, kể cả startup, phục hồi DB, và **khi connect as sysdba thì vào DB với user SYS** luôn (bất kể kết nối bằng user nào) [20] . Còn **SYSOPER** thì cho phép startup/shutdown nhưng không được truy cập dữ liệu user.

Thường chỉ DBA được cấp SYSDBA (thông qua **password file** bên ngoài DB). Ít khi dùng SYSOPER.

Dev: Vậy DBA làm gì để quản lý quyền mấy dev?

DBA: Thường tạo các role như anh nói. Ví dụ:
- Role `READ_ONLY_APP` chỉ có quyền select các bảng cần thiết.
- Role `RW_APP` cho phép select, insert, update, delete.

Sau đó grant role đó cho user ứng dụng. Khi dev yêu cầu thay đổi (ví dụ cần thêm quyền execute một proc), DBA sẽ cập nhật role rồi áp dụng. Cách này tập trung, sau kiểm tra audit cũng dễ (xem role có gì).

Dev: Hệ thống lớn chắc có trăm user, không có role chắc vỡ đầu.

DBA: Chuẩn.

Dev: Oracle có log lại ai làm gì không anh?

DBA: Có **audit** nếu bật. 12c có **Unified Auditing** – audit hầu hết mọi thứ (login, ai chạy lệnh gì trên đối tượng nào...). Cái này DBA cấu hình để theo dõi bảo mật.

Dev: Nghe căng ha, dev làm gì DBA biết hết .

DBA: (Cười) Ờ, "nhất cử nhất động" trong DB mà quan trọng là audit được. Tất nhiên không phải lúc nào cũng bật full, vì ghi nhiều log. Nhưng ở môi trường cần tuân thủ (compliance) như ngân hàng thì **bắt buộc** audit việc truy cập dữ liệu nhạy cảm.

Dev: Em hiểu.

DBA: Vậy tóm tắt phần này:
- **Nguyên tắc least privilege**: user chỉ được quyền tối thiểu cần dùng.
- Sử dụng **roles** để quản lý nhóm quyền cho tiện.

- Tránh dùng tài khoản quyền cao cho ứng dụng.
- Quản lý mật khẩu, profile chặt chẽ ở môi trường production.
- Biết các tài khoản đặc biệt (SYS, SYSTEM...) và bảo vệ chúng cẩn thận (mật khẩu mạnh, đổi định kỳ, không cho dev biết).

Dev: Vâng. Em hứa sau này làm DBA gặp dev nào xài user SYSTEM bừa bãi là em "chửi nhẹ" liền .

DBA: Haha, tốt.

Tối ưu hóa truy vấn SQL (Query Optimization)

Dev: Giờ tới phần em quan tâm nè: **tối ưu truy vấn**. Em viết SQL suốt nhưng thú thật ít khi nghĩ nhiều về hiệu năng, chỉ cần chạy đúng. Lên production query chậm lại đổ tại "database dở" .

DBA: Anh đoán được mà. Rất nhiều dev **chưa có tư duy tối ưu SQL**, dẫn tới code chạy tốt với data nhỏ nhưng lên data lớn thì **lẹt đẹt**. Tối ưu SQL là kỹ năng quan trọng cả với dev và DBA.

Dev: Oracle có **công cụ tối ưu (optimizer)** đúng không anh?

DBA: Đúng, Oracle dùng **Cost-Based Optimizer (CBO)** – nó tính toán **cost (chi phí)** của các phương án thực thi một câu SQL rồi chọn cách tối ưu nhất. CBO hoạt động dựa trên **thống kê (statistics)** về dữ liệu: số lượng bản ghi, độ phân bố giá trị, v.v.

Dev: Nghĩa là nếu thống kê sai hoặc thiếu thì optimizer chọn plan tệ?

DBA: Chính xác. **DBA phải đảm bảo cập nhật thống kê** thường xuyên cho bảng, index. Oracle có job tự động gather stats ban đêm. Nhưng khi data thay đổi đột biến, dev/DBA nên chủ động gather stats ngay.

Dev: Em nhớ có lệnh `DBMS_STATS.GATHER_TABLE_STATS`...

DBA: Đúng rồi.

Giờ nói về tối ưu truy vấn: Anh liệt kê vài **lỗi phổ biến** và best practice nhé:

- **Thiếu chỉ mục (index):** Dev tạo bảng xong insert data nhưng không tạo index, kết quả SELECT truy vấn theo điều kiện chậm do phải quét toàn bộ bảng (**Full Table Scan**). Quy tắc: **cột nào thường dùng để lọc (WHERE), join, hoặc sắp xếp thì xem xét tạo index** [21] [22]. Không phải tạo bừa mọi cột, nhưng thiếu index thì truy vấn lớn sẽ rất chậm.

VD: Bảng nhân viên 1 triệu dòng, query `WHERE NAME = 'John'` mà cột NAME không index -> Oracle phải đọc cả triệu dòng để tìm John, mất thời gian. Thêm index NAME, Oracle nhảy thẳng tới các dòng tên John, nhanh hơn nhiều [23].

- **Lạm dụng SELECT** :* Viết `SELECT *` thay vì chỉ chọn cột cần dùng. Cái này tệ ở chỗ:
- Đọc và chuyển nhiều dữ liệu không cần thiết (tốn I/O, mạng).
- Không tận dụng được index chỉ chứa các cột cần nếu select *.

- Ảnh hưởng kế hoạch thực thi (Oracle có thể chỉ dùng index để trả về kết quả nếu chỉ cần cột trong index – gọi là "index-only access").

Lời khuyên: Chỉ select những cột thực sự cần.

- **Không dùng điều kiện (WHERE) hoặc điều kiện không tối ưu:** Một số dev viết query thiếu filter, hoặc filter không theo index. Ví dụ:
- Quên điều kiện join, dẫn tới kết quả cartesian (nhân Descartes) cực lớn.

- Dùng **hàm trên cột** trong điều kiện. Ví dụ `WHERE UPPER(name) = 'JOHN'` trong khi name có index. Hàm UPPER làm index vô hiệu, Oracle phải full scan [24]. Nên tránh, hoặc tạo **function-based index** nếu cần.

- Dùng **wildcard đầu chuỗi**: `WHERE name LIKE '%Smith'` cũng không xài được index (vì wildcard đầu).

- **Không sử dụng bind variables:** Đây là lỗi kinh điển ở dev. Nếu code ứng dụng ghép giá trị trực tiếp vào SQL (ví dụ xây chuỗi SQL trong loop), Oracle sẽ coi mỗi câu là mới và phải *parse & optimize lại từ đầu*, tốn CPU và bộ nhớ. Dùng **bind variable** (SQL có tham số) sẽ cho phép Oracle tái sử dụng kế hoạch thực thi, **giảm đáng kể thời gian phân tích và CPU** [25] [26].

VD: Thay vì trong Java nối chuỗi `"...WHERE id = " + someId`, hãy dùng `PreparedStatement` với `?` – đó là bind. Oracle khi gặp câu query giống nhau khác giá trị bind, nó xài lại kế hoạch cũ, nhanh hơn nhiều. **Dev tuyệt đối không nên ghép chuỗi SQL bừa bãi**, vừa chậm vừa dễ dính SQL injection.

- **N+1 query (truy vấn lặp):** Cái này thuộc phần ứng dụng nhưng ảnh hưởng DB: dev viết code duyệt từng bản ghi rồi query DB cho từng bản ghi -> số truy vấn tăng tuyến tính. VD 1000 bản ghi thì 1000 query. DBA rất "dị ứng" kiểu này. Nên xử lý gộp trên SQL (JOIN hoặc IN với danh sách) thay vì gọi DB liên tục.

- **Không kiểm tra execution plan:** Dev ít khi xem **kế hoạch thực thi (EXPLAIN PLAN)** nên không biết truy vấn chạy kiểu gì. DBA khi tối ưu luôn xem plan để biết Oracle đang làm gì: full scan, index scan, join theo phương pháp nested loop hay hash join, chi phí bao nhiêu... Em nên học cách đọc execution plan.

Dev: Execution plan ra mấy con số và step khó hiểu lắm.

DBA: Ban đầu vậy, nhưng quen sẽ thấy nó cực kỳ hữu ích. Oracle cho xem plan bằng lệnh:

```
EXPLAIN PLAN FOR <query>;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Plan sẽ hiển thị dạng cây. Em chú ý các chỗ:
- **Full Table Scan** (TABLE ACCESS FULL): nếu bảng lớn mà thấy dòng này => có thể thiếu index.
- **Index Range Scan**: tốt, đang dùng index để lọc.
- **JOIN method**: Nested Loops (tốt cho ít dữ liệu), Hash Join (tốt cho nhiều dữ liệu), Merge Join,...
- **Cost**: con số Oracle ước tính, tương đối thôi nhưng cost lớn nhỏ cũng gợi ý.

Nếu thấy plan bất thường (ví dụ đáng lẽ dùng index mà lại full scan), có thể do stats không chính xác hoặc query viết chưa tối ưu.

Dev: Em sẽ tập dùng.

DBA: Ngoài ra, Oracle cung cấp công cụ **SQL Tuning Advisor** và **SQL Trace (event 10046)** để phân tích sâu, nhưng phức tạp nên tạm thời em nắm nguyên tắc chính.

Dev: Dạ.

DBA: Best practices tối ưu SQL cho Oracle, anh tóm tắt:

- **Dùng index hiệu quả:** tạo index hợp lý, tránh thiếu index ở cột lọc quan trọng [27].
- **Tránh thao tác làm mất tác dụng index:** như dùng hàm trên cột, toán tử LIKE '%x' ở đầu chuỗi [24].
- **Chỉ truy xuất dữ liệu cần thiết:** chọn cụ thể cột, giới hạn tập kết quả bằng WHERE, LIMIT (Oracle dùng ROWNUM hoặc 12c có FETCH FIRST N ROWS).
- **Dùng bind variables:** để tái sử dụng plan, giảm overload parse [25] (và ngừa SQL injection nữa).
- **Cân nhắc partition cho bảng lớn:** Partitioning là tính năng cho phép chia bảng thành nhiều phần (theo range, list, etc.). Query sẽ chỉ scan partition liên quan thay vì cả bảng, **cải thiện hiệu năng rất nhiều với dữ liệu lớn** [28] [29]. Tuy nhiên partition là tùy chọn Enterprise, phải có license và cấu hình phức tạp hơn, em biết để sau dùng.

- **Materialized View cho báo cáo phức tạp:** Với truy vấn tổng hợp nặng (SUM, JOIN nhiều bảng) lặp lại, DBA có thể tạo *materialized view* lưu sẵn kết quả tính toán. Người dùng query vào đó sẽ rất nhanh [30] [31]. Đổi lại tốn không gian lưu kết quả và phải refresh định kỳ.

- **Kiểm tra AWR để tìm SQL chậm:** AWR (Automatic Workload Repository) lưu lại các SQL "ngốn" tài nguyên nhất. DBA có thể xem **Top SQL** (dùng nhiều CPU nhất, đọc nhiều I/O nhất) rồi tập trung tối ưu chúng trước [32] [33]. Thường 20% SQL nặng chiếm 80% tải hệ thống – tối ưu mấy cái đó là thấy hiệu quả liền.

Dev: AWR chắc lát nữa anh nói kỹ hơn ở phần hiệu năng tổng thể.

DBA: Ừ.

Dev: Cho em hỏi: Có khi nào **tối ưu SQL theo hướng hạ tầng** không? Ý em là thay vì sửa SQL thì tăng RAM, thêm CPU để nó chạy nhanh hơn?

DBA: (Mỉm cười) Đó cũng là cách nhưng không nên lạm dụng. **Scaling up phần cứng** giúp nếu hệ thống bị thiếu tài nguyên, nhưng **SQL tồi** thì thêm bao nhiêu phần cứng cũng đốt hết. Anh luôn ưu tiên tối ưu logic SQL và index trước. Phần cứng là hỗ trợ.

Dev: Dạ.

DBA: Cuối phần này, anh "chửi nhẹ" dev tí cho nhớ:

- **"Đừng bao giờ** viết SQL mà không thèm nghĩ đến index hay plan. Viết xong ít nhất cũng chạy explain plan coi nó có **đi đường quyền** (full scan) gì không. Nếu có, xem lại liền."

- **"Đừng có dại** mà loop 1000 lần gọi DB rồi đòi nhanh. Đến máy thần cũng chịu. Hãy **tận dụng sức mạnh SQL** để xử lý dữ liệu theo set."

- **"SQL chạy chậm, trước tiên** tự nhìn lại mình **xem đã tạo index chưa, query viết ngu ở đâu chưa,** đừng vội đổ thừa **database."**

Dev: Trời, anh gằn giọng mà thấm thiệt. Em hứa từ nay viết SQL sẽ cẩn thận.

DBA: Tốt. Vậy chuyển sang bức tranh rộng hơn: **điều chỉnh hiệu năng toàn hệ thống** nhé.

Điều chỉnh hiệu năng tổng thể (Performance Tuning)

Dev: Phần này chắc nặng đây – ngoài tối ưu từng query, DBA còn phải tối ưu cả hệ thống DB.

DBA: Đúng. Nếu ví Oracle DB như một cái xe, viết SQL tốt là người lái giỏi, còn hiệu chỉnh hệ thống là tinh chỉnh máy móc, bảo dưỡng cho xe chạy khỏe.

Dev: Những gì anh phải làm trong mảng này?

DBA: Rất nhiều, nhưng anh sẽ gom thành vài ý chính: **theo dõi và tối ưu các tài nguyên chính: CPU, memory, I/O (đĩa), network**; và **quản lý tranh chấp tài nguyên (locking, latching)**.

1. Giám sát hiệu năng:

DBA dùng các **công cụ giám sát** như **AWR (Automatic Workload Repository)**, **ADDM (Automatic Diagnostic Monitor)**, **Enterprise Manager** để nắm tình trạng DB.

- **AWR báo cáo** theo chu kỳ (mặc định snapshot mỗi giờ): cho biết trong khoảng thời gian đó, DB dùng bao nhiêu CPU, I/O, top SQL nào tốn tài nguyên, các **điểm nghẽn (wait events)** nào nổi bật. Ví dụ xem AWR thấy phần **"SQL ordered by Elapsed Time"** có câu query XYZ chạy 5 phút chiếm 30% thời gian => tập trung tối ưu query đó.

- **Wait Events:** Oracle hoạt động theo cơ chế nếu process phải chờ tài nguyên gì sẽ ghi nhận "wait event". Ví dụ **DB file sequential read** (chờ đọc đĩa ngẫu nhiên – thường do index lookup), **DB file scattered read** (chờ đọc đĩa full scan), **enq: TX - row lock contention** (chờ khóa hàng do transaction khác giữ), v.v. DBA nhìn **Top 5 wait events** để biết hệ thống đang bị nghẽn chỗ nào nhất [34] [35].

- Nếu chủ yếu wait **đĩa I/O** -> có thể do thiếu index hoặc thiếu cache.

- Nếu wait **CPU cao** (CPU usage 100%) -> do nhiều truy vấn nặng CPU, cần tối ưu SQL hoặc tăng CPU.

- Nếu wait **lock** -> có vấn đề transaction concurency, phải tìm xem ứng dụng có chỗ nào khóa lâu (có thể do chưa commit kịp).

- **Real-time monitoring:** Dùng Enterprise Manager (giao diện web) hoặc query v$session để xem **hiện tại** ai đang chạy gì, có session nào đang chờ khóa.

Tóm lại, DBA liên tục **theo dõi metrics**: CPU load, memory sử dụng (SGA hit ratio, PGA usage, swap), I/O throughput, và **thống kê hiệu năng** để nắm sức khỏe DB [36] [37].

Dev: Công phu thật.

DBA: Ừ, làm "bác sĩ hệ thống" mà.

2. Tinh chỉnh cấu hình:

Dựa trên giám sát, DBA **điều chỉnh tham số** và cấu hình để tối ưu. Ví dụ:

- **Bộ nhớ:** Điều chỉnh kích thước **SGA, PGA** phù hợp workload. Oracle có tự động nhưng DBA vẫn cần đặt đúng giới hạn. Nếu SGA quá nhỏ -> cache hit thấp, Oracle phải đọc đĩa nhiều => chậm. Tăng SGA giúp tăng cache nhưng nếu lố RAM vật lý lại gây swap -> còn hại hơn. Nên phải cân bằng.

- **CPU:** Nếu CPU luôn full 100% và đã tối ưu SQL mà vẫn không giảm, có thể xem xét **thêm CPU hoặc phân quyền** lại. Ví dụ dùng **Resource Manager** của Oracle để hạn chế nhóm session ít quan trọng xài quá nhiều CPU, ưu tiên tài nguyên cho nhóm quan trọng.

- **I/O & Storage:** Kiểm tra **cấu trúc lưu trữ**: dữ liệu, chỉ mục có nên tách ổ khác nhau không (Oracle có khái niệm **tablespace RAID** – ví dụ đưa index và table trên 2 ổ vật lý riêng để đọc song song). Ở cấp hạ tầng, phối hợp sysadmin dùng **SSD** cho những tablespace "nóng" chẳng hạn.

- **Các tham số tối ưu khác:** Oracle có hàng trăm tham số init ẩn/hiện. Ví dụ:

- `DB_FILE_MULTIBLOCK_READ_COUNT` – ảnh hưởng tốc độ đọc full scan.
- `LOG_BUFFER` – kích thước log buffer, nếu giao dịch nhiều có thể tăng.
- `PGA_AGGREGATE_TARGET` hoặc `PGA_AGGREGATE_LIMIT` – kiểm soát bộ nhớ cho sort/hash.
- `parallel_degree` – cho phép chạy song song.

Tinh chỉnh đòi hỏi hiểu sâu và test cẩn thận.

Dev: Có tool tự động khuyến nghị không anh?

DBA: Có **ADDM** – sau mỗi snapshot AWR nó phân tích và đề xuất. Ví dụ: "SQL X tốn CPU cao, khuyến nghị tạo index" hoặc "SGA kích thước nhỏ, db buffer cache hit rate thấp, nên tăng DB_CACHE_SIZE". DBA tham khảo rồi quyết định. Đừng mù quáng làm theo.

Dev: Hiểu.

3. Quản lý khoá và cạnh tranh tài nguyên**:**

Oracle cho phép **nhiều người dùng truy cập đồng thời**. Dev thì quen tư duy đơn luồng, lên DB đa người dễ xảy ra tranh chấp:

- **Khoá cấp ứng dụng (row lock):** Nếu hai transaction cố cập nhật cùng một hàng, cái sau sẽ **phải chờ khoá** của cái trước commit/rollback. Dev đôi khi quên, để transaction mở quá lâu -> giữ khóa, người khác chờ dài cổ. DBA phát hiện qua wait event **"enq: TX - row lock contention"** rồi báo dev xử lý (ví dụ: "Em ơi code chỗ này sao mở transaction mà không thấy commit, khóa record suốt 5 phút rồi này!").

- **Deadlock:** Dev chạy hai giao dịch ngược nhau (A khóa bảng 1 chờ bảng 2, B khóa bảng 2 chờ bảng 1) -> deadlock. Oracle sẽ tự phát hiện và hủy 1, nhưng cũng gây gián đoạn. DBA xem **trace file** deadlock để báo dev fix code.

- **Latches, Mutexes:** Đây là khóa nội bộ trên tài nguyên memory (như cache). Thường nếu contention cao (VD **"latch: shared pool"**), có thể do không dùng bind causing parse nhiều. DBA phải xử lý nguyên nhân gốc.

Dev: Nghe khá phức tạp, chắc cần kinh nghiệm nhiều.

DBA: Đúng, kinh nghiệm trận mạc mới nhạy được.

4. Quy mô hệ thống và mở rộng (scaling):

- **Scale Up (Vertical):** nâng cấp máy chủ: thêm CPU, RAM, đĩa nhanh hơn. Oracle thường scale tốt lên đến giới hạn nào đó.

- **Scale Out (Horizontal):** nếu một máy không cáng nổi, có thể dùng **Oracle RAC (Real Application Clusters)** – chạy nhiều instance Oracle trên các nút khác nhau nhưng cùng truy cập một database (dùng chung bộ datafile). RAC cho phép tăng năng lực xử lý và dự phòng khi 1 node hỏng node khác vẫn chạy. Tuy nhiên RAC phức tạp và không phải lúc nào cũng hiệu quả (overhead phối hợp giữa các node).

- Hoặc dùng **Sharding** (12c cũng hỗ trợ): chia CSDL thành nhiều phần độc lập trên các máy khác nhau (mỗi shard một schema hoặc một range dữ liệu). Ứng dụng biết để phân phối truy vấn đến shard tương ứng. Cách này giống chia để trị, nhưng thay đổi ứng dụng lớn.

Dev: Ở tầm dev em ít động đến mấy cái này, nhưng biết để hình dung bức tranh lớn.

DBA: Ừ.

5. Các công cụ hỗ trợ hiệu năng:

- **Oracle Enterprise Manager (OEM):** Giao diện web đồ họa, có bảng điều khiển hiển thị performance, biểu đồ, cảnh báo... Cho DBA cái nhìn trực quan và quản trị toàn diện (backup, tạo user... cũng trên đây được).

- **AWR & Statspack:** AWR như nói ở trên (tính năng có phí bản Enterprise). Bản Standard có thể dùng **Statspack** (xưa hơn) để thu thập tương tự.

- **V$ và DBA_* views:** Oracle có hàng trăm view trong data dictionary cung cấp thông tin tức thời: `v$session` (session đang chạy), `v$sysstat` (thống kê hệ thống), `v$system_event` (wait event tổng), `v$sql` (câu SQL trong library cache), `DBA_INDEXES`, `DBA_TABLES`... DBA xài rất nhiều để truy vấn thông tin. Ví dụ muốn biết index fragment thế nào, xem `DBA_INDEXES` cột BLEVEL (độ cao cây index), nếu >4 có thể xem xét rebuild.

- **TKPROF**: công cụ format file trace SQL, dùng khi mình 10046 trace để phân tích chi tiết từng bước SQL (thời gian parse, fetch, I/O...).

Dev: Quá nhiều thứ mới để học.

DBA: Sẽ từ từ, em đừng ngợp. Về nguyên tắc, **hiệu năng DB = tổng hòa của SQL + tài nguyên + thiết kế dữ liệu + config**. DBA phải tối ưu toàn diện.

Anh ví dụ một tình huống điển hình để em hình dung cách nghĩ DBA nhé:

Tình huống: Hệ thống chậm bất thường.

- Việc đầu tiên DBA làm: mở **AWR report** hoặc xem **OEM Performance** trong khoảng thời gian chậm. Thấy **CPU 90%**, Top waits = "CPU Thủ tục" (meaning CPU-bound) và vài SQL chiếm nhiều CPU.

- Tiếp theo: Kiểm tra **Top SQL** – thấy câu `SELECT * FROM orders` không filter gì chạy nhiều lần, cost rất cao. Hỏi ra thì dev vừa release tính năng mới quét toàn bộ đơn hàng. Rõ ràng **SQL này vấn đề**.

- Hướng xử lý: Yêu cầu dev thêm điều kiện hoặc phân trang, không thể để query full dữ liệu hoài. Tạm thời, DBA có thể **thêm index** hỗ trợ cho bớt load, nhưng giải pháp gốc vẫn là sửa code.

- Ngoài ra, do CPU cao, DBA cũng xem xét **thêm CPU** cho máy nếu có thể, nhưng đó là plan B.

Dev: Em thấy pattern: DBA luôn tìm **nguyên nhân gốc** (root cause) chứ không chỉ chữa cháy bề mặt.

DBA: Chính xác. Giống bác sĩ phải tìm bệnh, không chỉ cho thuốc giảm đau.

Dev: Phần này chắc em phải thực hành nhiều mới thấm.

DBA: Ừ. Tóm tắt vài **lời khuyên thực chiến** về hiệu năng cho em:

- **Hiểu rõ workload**: biết hệ thống của mình kiểu OLTP (nhiều giao dịch nhỏ) hay OLAP (ít query nhưng nặng) để tối ưu phù hợp. OLTP thì ưu tiên index cho truy cập nhanh, OLAP thì tune cho các query lớn, có thể dùng partition, parallel query.

- **Proactive monitoring**: **đừng đợi người dùng kêu chậm mới đi tìm**, hãy theo dõi thường xuyên (qua biểu đồ OEM hoặc báo cáo định kỳ) để phát hiện xu hướng xấu (CPU tăng dần, query lâu hơn dần, v.v.) mà xử lý trước. "Phòng bệnh hơn chữa bệnh".

- **Minimize contention**: làm việc với team dev để cải thiện logic tránh khóa nhiều. Cấu hình các tham số, chế độ isolation level phù hợp (Oracle mặc định read committed tốt rồi, dev không nên dùng serializable nếu không cần).

- **Không chạy đua vũ trang vô ích**: Có câu **"There is nothing so useless as doing efficiently that which should not be done at all."** – làm việc vô ích dù nhanh cũng vẫn vô ích [38]. Hiểu là: Tối ưu nhất là **loại bỏ công việc thừa**. Ví dụ thay vì cố tăng tốc query báo cáo chạy hàng giờ, hỏi lại mới biết báo cáo đó **chẳng ai dùng**, tốt nhất ngừng chạy nó luôn cho khỏi tốn tài nguyên. DBA nên để ý những tác vụ vô ích để tư vấn bỏ đi.

Dev: Nghe triết lý quá, nhưng đúng thật .

DBA: Đó là chút kinh nghiệm anh đúc kết.

Sao lưu và Phục hồi (Backup & Recovery)

Dev: Tới phần không kém quan trọng: **backup**. Em đoán đây là "job số 1" của DBA đúng không?

DBA: Chính xác, **"Backup is Job #1"** – đảm bảo dữ liệu **không bao giờ mất** [39]. Một DBA có thể không giỏi tuning lắm nhưng **nhất định** phải vững phần backup, vì chỉ cần mất dữ liệu một lần thôi là... ra đường đấy em ạ.

Dev: (Nuốt nước bọt) Căng vậy anh.

DBA: Thật mà. Người ta có câu: **"DBA không bao giờ ngủ, chỉ nghỉ giữa các lần backup."**

Dev: Vậy anh cho em biết Oracle có những cách backup nào, và DBA làm hàng ngày ra sao?

DBA: Oracle hỗ trợ hai kiểu backup chính: **backup vật lý** và **backup lôgic**.

- **Backup vật lý:** sao lưu các file vật lý của database (datafile, controlfile, archived redo logs).
- **Backup lôgic:** xuất dữ liệu ở mức logic (bảng, schema) ra file (thường là dump file chứa các lệnh SQL để tạo lại dữ liệu).

Chúng tương ứng với công cụ:
- **RMAN (Recovery Manager):** công cụ chính thức của Oracle để backup vật lý.
- **Data Pump Export/Import (expdp/impdp):** công cụ để backup logic (xuất khẩu dữ liệu).

Dev: Anh nói chi tiết hơn về **RMAN** đi.

DBA: Được. **RMAN** là một chương trình (giao diện dòng lệnh) rất mạnh:

- RMAN kết nối đến database (target) với quyền SYSDBA, cho phép **sao lưu khi DB đang mở (online)** nếu DB chạy ở **ARCHIVELOG mode**.

- **ARCHIVELOG mode**: chế độ ghi lại toàn bộ giao dịch ra **archived redo logs** để có thể phục hồi tới từng thời điểm. Oracle DB khi cài thường chọn archivelog cho môi trường prod. Nếu chạy **NOARCHIVELOG** thì chỉ backup được khi DB shutdown (cold backup) – không phù hợp prod.

- RMAN có thể backup **full** (toàn bộ) hoặc **incremental** (chỉ phần thay đổi so với backup trước), backup **datafile riêng lẻ**, backup **archivelog** v.v.

- RMAN quản lý **catalog** ghi lại bản backup, hỗ trợ **restore** rất linh hoạt: có thể restore toàn bộ DB, hoặc từng tablespace, thậm chí từng block bị lỗi.

- RMAN cũng cho phép **duplication** (nhân bản DB từ backup) – hay dùng để clone môi trường test.

Dev: Nghe xịn thật.

DBA: Cực xịn. Ngày xưa DBA phải tự viết script backup file, giờ hầu hết dùng RMAN cho an toàn.

Ví dụ lệnh backup:

```
BACKUP DATABASE PLUS ARCHIVELOG;
```

Lệnh này tạo bản backup full DB và kèm các archive log.

Dev: Còn phục hồi thì sao?

DBA: Phục hồi (restore & recover) cũng qua RMAN:

- **Restore**: lấy file backup từ storage trả về đĩa.
- **Recover**: áp dụng các redo (archive log) để đưa DB về thời điểm mong muốn.

VD:

```
RESTORE DATABASE;
RECOVER DATABASE UNTIL TIME "TO_DATE('2025-07-01 08:00','YYYY-MM-DD
HH24:MI')";
```

Khôi phục DB tới 8h sáng 1/7/2025 (giả sử trước đó có backup và log đầy đủ).

Dev: Tức là có thể **phục hồi point-in-time**.

DBA: Đúng, đó gọi là **PITR (Point-In-Time Recovery)**. Miễn có log, em tua được.

Dev: Nếu lỡ xóa nhầm một bản ghi 1 tiếng trước, có cách nào phục hồi nhanh không anh?

DBA: Oracle có tính năng **Flashback** (nếu bật). Cho phép xem dữ liệu quá khứ (Flashback Query), phục hồi bảng về trạng thái quá khứ (Flashback Table), thậm chí phục hồi toàn DB về thời điểm trước (Flashback Database) mà không cần apply từng log. Nhưng config flashback khá phức tạp, tạm thời em cứ hiểu backup truyền thống đã.

Dev: Dạ.

DBA: Về **backup logic (Data Pump)**:

- Dùng lệnh `expdp` (Export Data Pump) để xuất dữ liệu ra file .DMP. Có thể xuất toàn bộ DB, hoặc chỉ schema, hoặc chỉ một số table.
- Lệnh `impdp` để import file .DMP vào DB (phục hồi dữ liệu).

Cách này thuận tiện để **di chuyển dữ liệu** giữa các DB (ví dụ chuyển từ Oracle này sang Oracle khác, khác phiên bản cũng được Data Pump xử lý).

Dev: Data Pump có coi là backup không?

DBA: Cũng là một dạng backup logic, nhưng trong khái niệm backup chính thống, người ta ưu tiên backup vật lý vì nhanh và đầy đủ. Data Pump chậm hơn và file dump cũng khá lớn, nhưng được cái linh hoạt (phục hồi chọn lọc đối tượng dễ).

Dev: Vậy trong thực tế, anh **lên lịch backup** thế nào?

DBA: Thường thì:
- **Hàng ngày**: backup incremental (nếu dùng) + backup toàn bộ archive log.
- **Hàng tuần**: backup full.
- Luôn chạy ở archivelog mode để có thể PITR.
- Giữ backup luân phiên 1-2 tuần, tùy chính sách.
- Backup nên lưu **ra nơi khác** (băng, NAS) phòng máy chủ hỏng hẳn.

DBA sẽ viết script hoặc dùng OEM thiết lập lịch chạy RMAN, sau đó **kiểm tra log** hàng ngày xem backup có lỗi không.

Dev: Nếu backup lỗi mà không biết thì nguy hiểm thật .

DBA: Ừ, **"Đừng bao giờ tin cấu hình backup chưa được** kiểm thử**"** [39]. Nghĩa là backup xong phải thỉnh thoảng thử **restore** xem có dùng được không. Nhiều trường hợp tới lúc cần phục hồi mới tá hỏa backup đã hỏng từ lâu.

Dev: Vãi.

DBA: Nên anh mới nói backup là job #1.
Câu cửa miệng DBA: **"Một bản backup chưa được thử restore thì không phải backup."**

Dev: Thâm thúy ghê.

DBA: Ngoài ra, cần tính đến **tình huống xấu**:
- Máy chủ hỏng hoàn toàn -> có backup offsite + tài liệu quy trình dựng lại server + restore DB.
- Lỗi do con người (như dev chạy lệnh xóa nhầm dữ liệu) -> DBA có thể dùng **recover tới thời điểm trước khi xóa** (PITR), hoặc nếu có flashback thì càng dễ.

    - Thảm họa (thiên tai...) -> nếu critical thì phải có **Data Guard** (Oracle Data Guard – CSDL dự phòng ở site khác luôn cập nhật).

Dev: Data Guard là dạng mirror hả anh?

DBA: Gần như. Nó apply liên tục archive log sang database standby. Khi primary die, có thể failover standby thành primary nhanh.

Dev: Quay lại chuyện dev, bọn em cần lưu ý gì về backup không?

DBA: Dev thì ít khi trực tiếp làm backup (trừ phi startup). Nhưng **dev cần ý thức**:

- **Trước khi làm thay đổi lớn hoặc công việc nguy hiểm trên DB, hãy yêu cầu DBA backup** (hoặc ít nhất export dữ liệu liên quan). Anh từng thấy có bạn dev chạy update sai điều kiện, bay mất dữ liệu quý. Nếu trước đó cẩn thận nhờ DBA export bảng đó ra thì cứu dễ, còn không thì phải phục hồi cả DB về trước khi update, rất phiền phức [40].

- **Đừng tự tiện "thử lệnh nguy hiểm" trên production**. Nhiều dev tò mò (hoặc nóng vội) chạy mấy lệnh DDL lớn không thông qua DBA, lỡ lỗi là ảnh hưởng rộng. Hãy để DBA làm hoặc giám sát.

- **Test khôi phục**: nếu dev phụ trách môi trường dev/test, cũng nên tập dượt restore từ backup để biết quy trình, tránh đến lúc cần thì lúng túng.

Dev: Anh có thể kể ví dụ thực tế nào cho sống động?

DBA: Ok. Ví dụ vui:

Một ngày đẹp trời, cô developer chạy lệnh xóa user trong ứng dụng nhưng quên thêm điều kiện "WHERE id=...", kết quả xóa toàn bộ user trong bảng. Cô tái mặt chạy qua báo DBA:

Dev: Anh ơi cứu em, em lỡ xóa hết data bảng Users rồi!!!

DBA: (Bình tĩnh) Có backup tối qua, mình có hai cách:
- Dùng RMAN phục hồi bảng đó từ backup (12c có feature **Recover Table**).
- Hoặc import từ bản export logic mới nhất.

Trong trường hợp này, DBA chọn **Recover Table** bằng RMAN: đưa bảng Users về trạng thái trước thời điểm xóa. Sau 15 phút, data trả lại, dev thở phào.

DBA nói nhẹ: *"Lần sau nghịch dại thì báo anh export bảng ra trước nhé, đỡ hú hồn."*

Cô dev tạ ơn rối rít.

Dev: May quá có backup ha anh.

DBA: Đó, DBA luôn phải chuẩn bị cho tình huống xấu nhất.

Dev: Em rút ra: Thứ nhất, **luôn backup và** kiểm tra backup****. Thứ hai, dev làm gì cũng cần nghĩ tới phương án khôi phục nếu sai.

DBA: Chuẩn.

Dev: Nói nhỏ anh nghe, trước giờ dev bọn em ngại động đến backup lắm, toàn trông chờ DBA. Giờ nghe anh giảng mới thấy **DBA như người hùng thầm lặng** – lo hết những thứ xấu nhất có thể xảy ra .

DBA: Haha, công việc nó thế. Nhiều khi tụi anh làm tốt thì chẳng ai để ý (vì không có sự cố), nhưng chỉ cần lỗi backup hay chậm trễ là ăn chửi liền. Nhưng không sao, quen rồi.

Dev: Em hứa sau này làm DBA sẽ không để sự cố nào qua mắt mình (nói cứng vậy thôi chứ chắc cũng run).

DBA: Có tâm thế vậy là tốt.

Sự khác biệt trong tư duy Dev vs DBA

Dev: Anh ơi, nãy giờ qua các phần em đã thấy le lói sự khác biệt giữa dev và DBA. Anh có thể tổng kết rõ hơn không? Để em tự soi xem cần điều chỉnh tư duy thế nào.

DBA: Được thôi. Anh sẽ liệt kê vài khía cạnh nhé:

- **Mục tiêu công việc:** Dev tập trung vào **tính năng và thời gian giao hàng** (delivery). Quan trọng là **nhanh ra chức năng** đáp ứng yêu cầu nghiệp vụ. DBA tập trung vào **tính ổn định, an toàn và hiệu năng lâu dài** của dữ liệu.

  Developer: "Phải giao hệ thống với X nghìn dòng code cuối tháng này, tôi ưu tiên làm cho xong."
  DBA: "Dữ liệu là tài sản lâu dài. **Đừng vì deadline trước mắt mà hack bừa** schema hôm nay, sẽ phá hỏng giá trị dài hạn của dữ liệu" [15] .

- **Phạm vi quan tâm:** Dev thường quan tâm **module của mình** (scope hẹp), còn DBA phải quan tâm **toàn cục hệ thống DB**.
- Dev nghĩ: "Query của tôi chạy ổn trên dev, còn hệ thống chậm chắc không phải do tôi."
- DBA nghĩ: "Một query tệ có thể kéo chậm cả DB. Phải xem xét mọi thứ đang chạy."

DBA có cái nhìn **bao quát**: mỗi thay đổi đều cân nhắc ảnh hưởng đến backup, đến các ứng dụng khác, đến bảo mật... Dev nhiều khi chỉ nghĩ tính năng của app mình.

- **Thái độ với rủi ro:** Dev thường **ưa thay đổi, thử nghiệm** (văn hóa fail fast), còn DBA thì **thận trọng, sợ rủi ro**.
- Ví dụ deploy bản mới: Dev có thể bảo "Cứ đẩy lên, có gì sửa sau", DBA sẽ phản đối "Không, **phải test kỹ**, có kế hoạch rollback rõ ràng mới làm."

- Dev thích dùng công nghệ mới (ví dụ một tính năng mới của Oracle 12c) ngay, DBA hay bảo thủ đợi nó chín, vì sợ bug. (Có câu: "**Be not the first to try new, nor the last to leave old.**" – đừng chạy đầu cũng đừng quá trễ [41]).

- **Quản lý cấu hình & quy trình:** DBA sống trong thế giới quy trình chặt: môi trường riêng, ITIL, change management. Dev đôi khi thấy phiền phức quan liêu. Nhưng chính nhờ quy trình, DBA mới giảm thiểu sai sót khi quản trị dữ liệu.

- **Xử lý sự cố:** Khi lỗi xảy ra:

- Dev thường nghĩ ngay lỗi trong code mình (nếu stacktrace chỉ ra) và sửa code. Nếu lỗi mơ hồ, có thể **đổ tại môi trường** ("chắc do DB"), rồi ping DBA.

- DBA khi sự cố (ví dụ DB chậm) sẽ **xem logs, bằng chứng** rất cẩn thận, **tìm gốc rễ**. DBA giỏi thường bình tĩnh hơn, vì đã chuẩn bị kịch bản (có backup, có standby...).

- **Góc nhìn về dữ liệu:** Dev coi dữ liệu là phương tiện cho ứng dụng (thậm chí DB chỉ như "nơi chứa" – some devs say "database is just a dumb storage"). DBA thì **xem dữ liệu là trung tâm**, ứng dụng có thể thay đổi nhưng dữ liệu phải nhất quán, trường tồn.

- Vì vậy dev có thể code insert bừa (miễn chạy được), chứ DBA thì đặt nặng ràng buộc (constraints) đảm bảo dữ liệu đúng.

- **Kinh nghiệm đau thương:** Dev sợ nhất bug làm app crash. DBA sợ nhất *mất dữ liệu*. Dev fix bug xong có khi còn được khen, chứ DBA mà để mất dữ liệu dù cứu lại được cũng bị trách (ít ai khen "may nhờ backup" đâu).

- Như có người nói vui: *"Dev gây ra bug -> sửa bug -> được khen đã khắc phục. DBA mà gây ra lỗi -> sửa lỗi -> cũng chẳng ai khen, vì họ bảo đáng lẽ anh không được làm lỗi đó!"* [42] .

Dev: Haha, đúng là nghiệp DBA bạc hơn dev nhỉ.

DBA: Nên nhiều người **stress** đó em.

Dev: Em thấy dev dễ được nổi hơn, làm xong tính năng người dùng thấy ngay. DBA làm tốt thì chẳng ai biết vì hệ thống "ngon sẵn rồi".

DBA: Chuẩn.

Dev: Nhiều công ty chắc hai bên hay **mâu thuẫn** ha anh?

DBA: Có, kinh điển luôn: Dev thường nói DBA cổ hủ, cản trở ("Don't Bother Asking" = DBA ám chỉ mấy ông đừng có hỏi, cứ cản thôi). DBA thì nói dev ẩu, không hiểu biết. Thực ra hai bên **mục tiêu khác nhưng cùng chung đích**: cung cấp hệ thống ổn định, đáp ứng nghiệp vụ.

Dev: Vâng.

DBA: Cách tốt nhất là **hợp tác**. Những công ty làm tốt là dev và DBA làm việc từ đầu dự án: cùng thiết kế DB, cùng lên tiêu chuẩn coding, chỉ số hiệu năng, v.v. Dev hiểu hơn về DB, DBA hiểu nhu cầu dev –> suôn sẻ.

Dev: Em đồng ý. Em nghĩ dev muốn sang DBA như em cũng dễ vì hiểu bên kia.

DBA: Đúng, em có lợi thế hiểu được tâm lý dev, sau này làm DBA sẽ dễ thông cảm và hỗ trợ dev hơn, thay vì cãi nhau .

Dev: Từ nãy giờ anh cũng "chửi yêu" dev nhiều rồi, nhưng anh có thấy **dev cũng có cái khó** của họ không?

DBA: Có chứ. Nhiều khi dev bị áp lực thời gian, khách hàng đổi yêu cầu xoành xoạch, họ buộc phải *xoay xở* cho kịp, không còn thời gian tối ưu hay học sâu về DB. DBA nên hiểu điều đó để hỗ trợ, ví dụ:
- Thay vì quát "sao code ngu thế", hãy gợi ý giải pháp tốt hơn cho dev.
- Chủ động cung cấp cho dev environment gần prod (nhiều data) để họ test hiệu năng sớm.

Dev: Ghi nhận.

DBA: Anh cũng từng làm dev nên anh hiểu. Thật ra dev giỏi cũng nên học cách nghĩ của DBA để code xịn hơn.

Dev: Dạ, mục tiêu của em đây mà.

DBA: Vậy phần khác biệt tư duy anh tóm gọn:

- Dev **nhanh**, DBA **chắc**.
- Dev nghĩ **tính năng**, DBA nghĩ **dữ liệu**.
- Dev sợ **trễ deadline**, DBA sợ **mất dữ liệu**.
- Dev phạm lỗi thường do **thiếu kinh nghiệm hệ thống**, DBA phạm lỗi thường do **quy trình lỗi hoặc chủ quan**.

Dev: Nghe như triết lý Đông Tây vậy .

DBA: Haha, nhưng hiểu điểm này giúp em tự điều chỉnh khi chuyển vai trò.

Ví dụ tình huống thực tế Dev vs DBA

Dev: Anh kể thêm vài tình huống *cụ thể* nữa đi, cho em học kiểu scenario để nhớ lâu.

DBA: Được, anh có vài câu chuyện "dở khóc dở cười" đây:

Tình huống 1: Query chậm kinh khủng

Bối cảnh: Ứng dụng web mới triển khai, đêm đầu tiên chạy job tính báo cáo, hệ thống **đơ toàn tập**.

- **Dev hoảng hốt**: "Anh DBA ơi, server treo hay sao ấy, app em timeout hết!"
- **DBA kiểm tra**: thấy CPU 100%, ổ đĩa đọc liên tục. Trong **Top SQL** có một câu query từ job kia đang full-scan bảng 10 triệu dòng, join 5 bảng khác, **thiếu index**.
- **DBA hỏi dev:** Query này ai viết vậy em?
- **Dev:** Dạ code bên em, chạy ngon trên test (dữ liệu có 1000 dòng...). Giờ nhiều dữ liệu quá nên mới...
- **DBA:** (thở dài) Rồi, giờ tạm dừng job, anh tạo index giúp em trước, rồi chạy lại thử. Lần sau tính toán dữ liệu lớn thì phải nghĩ chuyện index, chia nhỏ, chứ viết kiểu này "giết" DB lắm.
- **Dev:** Dạ vâng, tại bọn em không ước lượng được dữ liệu thực.

Kết quả: Sau thêm index và tối ưu lại SQL (loại bỏ subquery thừa, thêm điều kiện phân đoạn), job chạy êm, server khỏe. Dev rút kinh nghiệm: **luôn test trên tập dữ liệu lớn và làm việc với DBA để tối ưu** trước khi lên prod.

Bài học: Query chạy nhanh với ít dữ liệu không đảm bảo gì cả. **Dev cần tư vấn DBA khi viết query phức tạp hoặc bảng rất lớn** để có giải pháp tối ưu (index, partition).

Tình huống 2: Chết vì thiếu COMMIT

Bối cảnh: Ứng dụng giao dịch tài chính. Bỗng một ngày **hàng loạt giao dịch treo**, người dùng complaint ầm ĩ.

- **DBA kiểm tra ngay**: thấy rất nhiều session **đang chờ khóa** trên bảng TRANSACTIONS. Ai đang giữ khóa? Một session duy nhất của batch process do dev viết, bắt đầu 30 phút trước, chưa commit cũng chưa rollback.

- **DBA gọi dev:** "Em chạy batch gì mà khóa bảng 30' rồi chưa commit?"

- **Dev (lo lắng):** "Dạ batch insert 100k bản ghi, em để cuối cùng mới commit một thể... Chắc nó đang chạy chưa xong?"
- **DBA:** "Chưa xong đâu, mà làm vậy là không ổn: **100k bản ghi để một transaction** vừa dễ khóa lâu, vừa nguy cơ undo, rollback lâu nếu fail. Em nên commit từng chunk, ví dụ mỗi 5k."
- **Dev:** "Em sợ commit nhiều ảnh hưởng ACID..."
- **DBA:** "Trong xử lý batch, chia nhỏ commit hợp lý không sao cả, miễn logic đảm bảo. Giờ tạm kill session đó cho user khác hết treo đã."

DBA kill session, nhưng do nó có 100k insert chưa commit, Oracle phải rollback -> mất thêm 10 phút DB vẫn load cao.

- **DBA (nhắc nhở):** Lần sau **đừng thiết kế batch khóa lâu vậy**, hoặc ít nhất chạy ngoài giờ. Em thấy đấy, suýt sập hệ thống.
- **Dev (xuống giọng):** Vâng, lần sau em sẽ commit từng phần, và thông báo trước cho anh khi chạy batch lớn.

Kết quả: Dev cập nhật batch job commit mỗi 1000 dòng, chạy ngon.

Bài học: Transaction ngắn vs dài: Dev thường chỉ nghĩ logic, quên rằng transaction lâu sẽ giữ khóa, gây **tắc nghẽn**. **DBA tư duy**: luôn tối thiểu hóa thời gian giữ khóa. Nên commit sớm khi có thể, đặc biệt trong batch processing.

Tình huống 3: "Em chỉ sửa tí dữ liệu thôi mà!"

Bối cảnh: Dev có quyền truy cập DB production ở mức hạn chế. Có bug dữ liệu nên sếp giục **sửa gấp** một số bản ghi. Dev chưa kịp xin DBA, nghĩ "mình có quyền update thì làm luôn cho nhanh".

Dev chạy một câu UPDATE không có điều kiện chặt -> lỡ sửa sai 5000 bản ghi khác.

- **Dev hoảng sợ**: ngắt query nhưng đã muộn, data lộn xộn. Lập tức thú nhận với DBA.
- **DBA:** (giận nhưng vẫn bình tĩnh) "Rồi, trước mắt cho ứng dụng dừng nhận giao dịch phần liên quan. May anh có backup logic (export) từ đêm qua. Để anh import tạm bảng đó ra schema khác rồi viết script đối chiếu khôi phục giá trị."

DBA dùng bản export đêm qua của bảng, so sánh với bảng hiện tại để tìm ra bản ghi lệch, sau đó phục hồi giá trị cũ. Mất 1 giờ căng thẳng nhưng cứu được.

- **DBA (nghiêm nghị):** Lần sau có yêu cầu sửa dữ liệu, **tuyệt đối phải qua anh**. Anh sẽ backup trước rồi mới cho làm. Em tự ý là suýt gây mất mát rồi.

- **Dev (cúi đầu):** Em xin lỗi, tại sếp hối quá em cuống lên...

- **DBA:** Sếp có hối thì càng phải làm đúng quy trình. Em thấy không, chút nữa là anh phải restore cả database luôn chứ đùa.

Bài học: Dev tuyệt đối không nên "tự xử" dữ liệu trên môi trường thật nếu không chắc chắn. DBA có quy trình an toàn hơn. Lỗi con người là khó tránh, nên phải *có lưới an toàn* (backup, kiểm tra chéo).

Tình huống 4: Mất cả chì lẫn chài vì... tên cột

Cái này vui: Một anh dev viết code .NET query Oracle, dùng câu SQL:

```
SELECT id, name, date
FROM users
```

Chạy trên Oracle bị lỗi. Lý do: **DATE** là từ khóa (kiểu dữ liệu) nên Oracle hiểu nhầm. Dev không biết, cứ tưởng Oracle lỗi thời không hiểu lệnh cơ bản. Cậu ta cáu và bảo DBA Oracle có vấn đề.

- **DBA**: (xem query) "Trời ạ, em đặt tên cột là `date` à? Phải để trong dấu ngoặc kép hoặc đổi tên khác. Đây không phải lỗi Oracle."
- **Dev:** "Ơ bên SQL Server em đặt `date` vô tư có sao đâu."
- **DBA:** "Khác hệ quản trị là khác chuẩn đấy em. Dùng từ khóa làm tên cột luôn là practice tệ. Đổi tên cột thành created_date hoặc gì đi."

Dev ngượng, sau đó đổi tên cột.

Bài học: Hiểu biết khác biệt ngôn ngữ SQL giữa các hệ DB cũng quan trọng. Dev hay assume cái gì bên này chạy được bên kia cũng được – không phải. Nên khi chuyển sang Oracle, dev phải học chuẩn SQL của Oracle (một số hàm, từ khóa khác).

Dev: Haha câu chuyện tên cột này em cũng từng dính.

DBA: Chuyển sang Oracle mà xài `dbo.table` hay `LIMIT` thay vì `ROWNUM/FETCH` cũng đầy, nên dev phải cập nhật kiến thức.

Công cụ và phương pháp tốt nhất cho Oracle DBA

Dev: Anh ơi, phần cuối chắc nói về các **công cụ** và **best practices** tổng hợp để em làm hành trang.

DBA: Đúng. Mình điểm lại và bổ sung vài thứ:

Công cụ quản trị và hỗ trợ:

- **SQL*Plus:** Công cụ dòng lệnh "huyền thoại" của Oracle. DBA phải biết dùng để chạy script, automate.

- **Oracle SQL Developer:** Công cụ GUI (miễn phí) hữu ích cho cả dev và DBA: kết nối xem schema, thiết kế query, xuất dữ liệu, v.v.

- **Oracle Enterprise Manager (OEM/Cloud Control):** Bộ quản trị tập trung, rất mạnh (nhưng bản đầy đủ tốn license). Có giao diện web theo dõi performance, thiết lập backup, cảnh báo threshold... Ở môi trường lớn gần như không thể thiếu.

- **RMAN:** Nhắc lại, công cụ backup/restore chính. Thường viết script `.rcv` để thực thi hàng đêm hoặc dùng OEM lên lịch.

- **Data Pump (expdp/impdp):** Đã nói, dụng cụ xuất/nhập dữ liệu. Dev/DBA đều có thể dùng để chuyển dữ liệu giữa các môi trường (ví dụ refresh dữ liệu từ prod -> test).

- **tkprof & trace utilities:** Dùng khi cần tối ưu sâu. Bật trace cho session hoặc toàn hệ thống để ghi lại từng câu lệnh, thời gian thực thi, rồi dùng tkprof format ra dễ đọc. Hữu ích để debug chỗ nào chậm.

- **AWR, ADDM, ASH reports:**

- AWR: báo cáo snapshot hiệu năng (có cả phần tư vấn của ADDM).

- ASH (Active Session History): liệt kê các session hoạt động trong khoảng thời gian, cho biết ai chạy gì, wait event nào – để phân tích chi tiết khi có sự cố tức thời.

- **OEM Performance Pages:** trong OEM có biểu đồ rất trực quan: xem **SQL Monitoring** (theo dõi real-time câu SQL dài đang chạy), **Instance Efficiency** (các tỷ lệ hit cache, CPU idle...), **Top Activity** (session tốn tài nguyên theo thời gian). DBA trực màn hình này có thể phát hiện bất thường ngay.

- **Toad, PL/SQL Developer:** Các công cụ bên thứ ba (có phí) giúp viết và debug SQL, PL/SQL tốt hơn, giao diện thân thiện. Nhiều DBA dùng Toad cho nhanh thay vì SQL Developer.

- **Scripting & Automation:** DBA thường xài Python, Bash, PowerShell... để tự động hóa (vd: script kiểm tra không gian, script gửi mail cảnh báo). Biết chút lập trình giúp ích nhiều.

Các phương pháp và thói quen tốt:

- **Quản lý cấu hình DB:** Lưu lại các tham số cấu hình (init.ora), lịch sử thay đổi. Mỗi lần điều chỉnh gì phải ghi chú (vì vài tháng sau quên sẽ khó truy vết tại sao set thế).

- **Version control cho schema:** Nên áp dụng quản lý phiên bản cho các script tạo bảng, alter bảng... giống code. Dev DBA phối hợp dùng git, mỗi thay đổi DB schema có script rõ ràng. Không làm kiểu "ai nhớ gì làm nấy".

- **Tài liệu vận hành:** Có tài liệu các quy trình quan trọng: cách restore, cách thêm node RAC, cách xử lý khi archive log đầy... Lúc sự cố đỡ cuống.

- **Bảo mật & cập nhật:** Thường xuyên cập nhật **patch** cho Oracle (mỗi quý có CPU – Critical Patch Update). Nhiều dev không để ý, nhưng DBA phải lên kế hoạch patch để vá lỗi bảo mật, nâng cao ổn định. Oracle nổi tiếng bảo mật tốt, nhưng nếu không patch thì lỗ hổng cũng nguy.

- **Kiểm tra định kỳ:**

- Check **alert log hàng ngày** xem có lỗi ORA- nào không.
- Check **backup logs** hàng ngày.
- Check không gian đĩa (đặc biệt ổ chứa archive log, vì nếu đầy archive log sẽ làm DB tê liệt).

- Test khôi phục (ví dụ thi thoảng thử restore backup trên máy test).

- **Luôn có phương án DR (Disaster Recovery):** Bên cạnh backup tại chỗ, nếu hệ thống quan trọng, phải có ít nhất backup offsite, tốt hơn là Data Guard. Test DR định kỳ (giả lập mất data center chính, bật site dự phòng).

- **Tối ưu thiết kế từ đầu:** Tham gia từ giai đoạn thiết kế DB:

- Đề xuất **chuẩn đặt tên** (đồng nhất, có tiền tố tiền tố, viết hoa hay thường, v.v.).
- Thiết kế **cấu trúc bảng, chỉ mục** chuẩn ngay từ đầu (có normalize, có index cho khóa chính, khóa ngoại, v.v.).
- Cân nhắc **partitioning** nếu bảng dự tính rất lớn.

- Tích hợp **monitoring** ngay (ví dụ triggers audit, hoặc thống kê ghi log ứng dụng).

- **Học và cập nhật:** Công nghệ DB cũng thay đổi. Một DBA tốt luôn cập nhật tính năng mới (ví dụ 12c có Adaptive Execution Plans – tự đổi plan giữa chừng, hay Memoptimized Rowstore...), học hỏi từ cộng đồng (các diễn đàn Oracle, blog, tài liệu Oracle).

- **Kỹ năng mềm:** Giao tiếp với dev, quản lý. Giải thích cho dev hiểu tại sao mình yêu cầu cái này, hướng dẫn họ best practice DB (nhiều dev trẻ rất cần sự mentor của DBA thay vì chỉ phán "code chú dở quá").

Dev: Nhiều cái hay quá anh ạ. Đọc mà muốn thực hành liền.

DBA: Tốt.

Dev: À, một câu nữa, *"lời chửi thẳng thắn"* cuối buổi anh muốn gửi đến dev – những người đồng đội tương lai của em?

DBA: Có chứ, anh có nguyên **"cẩm nang chửi yêu"** đây:

- **"Đừng có lười!"** – Lười tìm hiểu DB, lười tối ưu, lười viết thêm chỉ mục => Hậu quả: hệ thống ì ạch, DBA vất vả dọn dẹp. Dev nên bỏ chút thời gian nghĩ đến database chứ đừng coi nó là hộp đen.

- **"Đừng làm bố thiên hạ!"** – Tức là *đừng nghĩ mình code ngon rồi muốn query gì DB phải chiều nấy*. Có mấy ông dev query chạy 10 phút xong bảo "DB bên anh yếu". Xin thưa, **viết query tồi thì DB xịn cỡ nào cũng gánh không nổi**. Tôn trọng tài nguyên chung.

- **"Sai thì nhận, đừng đổ lỗi!"** – Khi có sự cố, hãy cùng DBA phân tích. Dev thật thà "chết rồi code em có bug" còn sửa được, chứ kiểu giấu giấu hoặc khăng khăng "không phải do em" thì khó xử lý lắm. DBA cũng nên cởi mở, blame game không giải quyết vấn đề.

- **“Học SQL đi!”** – Ngôn ngữ chính để làm việc với DB mà nhiều dev hững hờ, chỉ dựa ORM. ORM tốt nhưng hiểu SQL lõi sẽ giúp dev làm chủ hiệu năng và tránh lỗi ngớ ngẩn. Nhất là **phải biết mấy thứ anh dạy ở trên: index, execution plan, transaction**…

- **“Tôn trọng dữ liệu!”** – Đừng nghĩ “mất tí dữ liệu không sao, nhập lại được”. Có những dữ liệu không thể tái tạo. Làm gì liên quan data phải thận trọng, test kỹ, backup trước. Sai một ly, đi một dặm – ở đây là bay màu sự nghiệp.

- **“Xin đừng tự ý làm DBA khi chưa đủ trình.”** – Dev đôi khi tự ái “DBA làm được mình cũng làm được” rồi tự ý táy máy config DB, phục hồi DB mà **không thông qua DBA**. Sai sót phát sinh có thể rất đắt giá. Hãy **phối hợp** thay vì tự làm tất.

Dev: Đúng là “chửi” thật mà thấm thía. Em nghe mà như anh đang nói với phiên bản dev ngày xưa của anh ấy .

DBA: (Cười lớn) Ừ, ngày xưa anh cũng dính vài cái trong đó chứ ai. Học từ sai lầm mới khôn ra.

Dev: Em cảm ơn anh rất nhiều. Qua cuộc nói chuyện dài này, em học được cực kỳ nhiều kiến thức bổ ích từ cơ bản đến nâng cao.

DBA: Không có gì. Anh tin nếu em chịu khó tìm hiểu và rèn luyện, em sẽ làm DBA tốt. Nhớ rằng: **kiến thức hàn lâm kết hợp kinh nghiệm thực tế** mới vững. Đừng ngại những khái niệm “cao siêu” – hiểu chúng để áp dụng khi cần, nhưng cũng đừng sa đà lý thuyết mà quên thực tiễn.

Dev: Dạ, em sẽ cố gắng.

DBA: Ok, **welcome to the DBA world!** Chắc chắn nhiều thử thách nhưng cũng rất thú vị.

Dev: Vâng, hy vọng sớm trở thành đồng nghiệp “ngầu” như anh .

DBA: (Vỗ vai) Cố lên!

Tài liệu tham khảo: Các khái niệm và hướng dẫn trên dựa theo tài liệu Oracle và kinh nghiệm thực tế: Multitenant Architecture [8] [1], quản lý tablespace [43], tối ưu SQL [23] [25], backup & recovery [39], phân quyền SYSDBA vs DBA [14], cùng nhiều nguồn uy tín khác. Chúc bạn học tốt và trở thành DBA thực thụ!

---

[1] [2] [3] [4] [5] [8] Getting started with Oracle Database 12c Multitenant Architecture - Simple Talk
https://www.red-gate.com/simple-talk/databases/oracle-databases/getting-started-with-oracle-database-12c-multitenant-architecture/

[6] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] Best Practices for Faster Queries: A SQL Performance Tuning Tutorial | HackerNoon
https://hackernoon.com/best-practices-for-faster-queries-a-sql-performance-tuning-tutorial

[7] Understanding SGA and PGA in Oracle: A Simple Guide - Medium
https://medium.com/@ibukunjeje/understanding-sga-and-pga-in-oracle-a-simple-guide-4f27539b88dc

9 10 11 12 Oracle Architecture Overview - Simple Talk
https://www.red-gate.com/simple-talk/databases/oracle-databases/oracle-architecture-overview/

13 14 20 permissions - How are the DBA, SYSDBA, SYSOPER roles and SYS, SYSTEM users related in Oracle? - Database Administrators Stack Exchange
https://dba.stackexchange.com/questions/287356/how-are-the-dba-sysdba-sysoper-roles-and-sys-system-users-related-in-oracle

15 DBAs vs Developers: A Sad Tale of Unnecessary Conflict - Simple Talk
https://www.red-gate.com/simple-talk/opinion/opinion-pieces/dbas-vs-developers-a-sad-tale-of-unnecessary-conflict/

16 38 39 40 41 Oracle DBA Best Practices Dennis Williams Senior Database Administrator Lifetouch, Inc. - ppt download
https://slideplayer.com/slide/5671434/

17 Exam 1z0-082 topic 1 question 8 discussion - ExamTopics
https://www.examtopics.com/discussions/oracle/view/22518-exam-1z0-082-topic-1-question-8-discussion/

18 3 Managing User Privileges - Database - Oracle Help Center
https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tdpsg/managing-user-privileges.html

19 Oracle User Management - Learnomate Technologies
https://learnomate.org/oracle-12c-user-management/

21 22 A poorly designed relational database can lead to performance bottlenecks, data inconsistencies, and scaling challenges. | Brightscout
https://www.linkedin.com/posts/brightscout_database-development-scalability-activity-7256704420338434048-nNSv

34 35 36 37 Oracle 12c Performance Tuning - Top Tips for DBAs - DNSstuff
https://www.dnsstuff.com/tuning-oracle

42 Evolute from a Developer to DBA - Ask TOM
https://asktom.oracle.com/ords/asktom.search?tag=evolute-from-a-developer-to-dba

43 Understanding Tablespace Privileges and Granting Permissions in Oracle Database | by Oz | Medium
https://ozwizard.medium.com/tablespace-previllages-and-grant-49706f47f34a