数据库课程设计视频讲解脚本

用途：按 `campus_hub_demo_slides.pptx` 录制视频。PPT 是主画面；代码只展示 `campus-hub-next/packages/db/final-ddl/course-design-final-schema.sql`；网页只展示运行验证结果。

标记说明：

- 【PPT】保持在 PPT 当前页讲。
- 【代码】切到 `course-design-final-schema.sql`，按提示搜索或跳转到对应行。
- 【网页】切到浏览器页面，展示运行结果。
- 【回 PPT】切回 PPT 继续下一页。

建议时长：10 到 12 分钟。录制时不逐行读代码，只说明这段数据库对象解决了什么设计问题。

0:00-0:40 第 1 页：平台基础整体数据库设计

【PPT：第 1 页】

老师好，我本次课程设计是校园生活平台。视频里我主要介绍数据库相关内容，不重点展开前端页面样式。

整个系统按三个数据库子模块讲：平台基础、功能房预约、课程资源分享。这里先从平台基础开始，因为它为后面两个业务模块提供统一的身份、组织、权限和审计底座。

平台基础里，`profiles` 是认证用户的业务资料扩展；`departments` 和 `department_closure` 处理组织层级；`roles`、`permissions` 和 `user_roles` 处理 RBAC 授权；`audit_logs` 记录关键操作事实。这里体现的数据库设计点包括一对一扩展、多对多桥接、层级数据建模、审计事实表、索引和触发器。

0:40-1:45 第 2 页：部门闭包表与层级查询

【PPT：第 2 页】

平台基础里我认为比较有代表性的设计是部门闭包表。部门本身是树结构，`departments.parent_id` 只能表达直接父子关系。如果要经常查询“某部门及其所有子部门”，只靠递归查询会比较麻烦，也不利于权限范围过滤。

所以这里额外设计了 `department_closure`，保存所有“祖先部门到后代部门”的可达关系。`depth = 0` 表示自己到自己，`depth = 1` 表示直接父子关系。这样查询某个部门的全部下级时，只需要按 `ancestor_id` 查询闭包表。

【代码：跳到 `course-design-final-schema.sql:133` 附近】

这里展示 `departments` 和 `department_closure`。重点看 `department_closure` 的三个字段：`ancestor_id`、`descendant_id`、`depth`，以及 `(ancestor_id, descendant_id)` 复合主键。这个复合主键保证一个祖先到一个后代只会有一条路径记录。

【代码：跳到 `course-design-final-schema.sql:512` 附近】

再展示闭包表的维护触发器。插入部门时，先插入自己到自己的路径；如果有父部门，再把父部门的所有祖先复制成新部门的祖先。移动部门时，触发器会先检查是否形成环，再删除旧路径、补充新路径。

这里对应数据库课程设计中的层级数据设计、触发器维护派生关系、复合主键和索引设计。

1:45-2:45 第 3 页：RBAC、资料同步与审计约束

【PPT：第 3 页】

这一页展示平台基础的另外三个设计点：用户资料同步、RBAC 授权桥接、审计日志只追加。

【代码：跳到 `course-design-final-schema.sql:223` 附近】

先看 `profiles`。它和 `auth.users` 共用主键，认证表负责登录，`profiles` 保存业务侧要展示和管理的学号、姓名、状态等字段。学号有格式检查，必须符合 16 位数字格式，同时还建立了唯一索引，避免重复学号。

【代码：跳到 `course-design-final-schema.sql:285` 附近】

然后看 `user_roles` 和 `role_permissions`。这两个表都是典型的多对多关系表，用复合主键防止重复授权。用户可以有多个角色，角色可以对应多个权限，这比把权限直接塞进用户表更符合关系模型。

【代码：跳到 `course-design-final-schema.sql:390` 附近】

最后看 `audit_logs`。它保存操作者、目标对象、操作类型、差异内容等审计事实。再跳到 `course-design-final-schema.sql:427` 附近，可以看到触发器函数会禁止更新和删除审计日志，也就是审计表只允许追加。这样关键操作能够被追溯。

2:45-3:25 第 4 页：平台基础运行验证

【PPT：第 4 页】

这一页是平台基础的运行验证。

【网页：打开 `/console/users`】

先看用户页面，可以看到用户资料、状态、学号和角色信息，验证 `profiles` 与认证用户同步后的业务资料能够正常使用。

【网页：打开 `/console/departments`】

再看部门页面，页面上的树形结构对应 `departments` 的直接父子关系，范围查询背后可以使用 `department_closure`。

【网页：打开 `/console/roles`】

最后看角色页面，角色、权限和数据范围可以配置和回显，对应 `user_roles`、`role_permissions` 以及数据范围相关表。这里的重点不是页面样式，而是数据库设计能够支撑身份、组织和权限管理。

【回 PPT：第 5 页】

3:25-4:05 第 5 页：功能房预约整体数据库设计

【PPT：第 5 页】

第二个模块是功能房预约。这个模块的数据库重点是有限资源的时间占用、预约参与人集合、审核状态和并发冲突控制。

整体关系上，`facility_rooms` 是房间资源主体；`facility_reservations` 是预约主表，保存房间、申请人、时间窗口、状态、审核字段和取消字段；`facility_reservation_participants` 是预约参与人联系表；`facility_bans` 用来保存封禁规则。

这里没有在预约表里重复保存房间名或楼房名，而是通过外键关联房间主数据，避免冗余带来的更新不一致。

4:05-5:10 第 6 页：时间冲突与并发控制

【PPT：第 6 页】

功能房预约最关键的规则是：同一个房间，在 `pending` 或 `approved` 状态下，时间区间不能重叠。两个区间重叠的判断是：已有开始时间小于新结束时间，并且已有结束时间大于新开始时间。

【代码：跳到 `course-design-final-schema.sql:879` 附近】

这里展示数据库层的排斥约束 `facility_reservations_room_active_time_excl`。它使用 `EXCLUDE USING gist`，把 `room_id` 相等和 `tstzrange(start_at, end_at, '[)')` 区间重叠组合起来限制。

这里的 `[)` 表示包含开始、不包含结束，所以 10 点到 11 点和 11 点到 12 点不冲突。约束只作用于 `pending` 和 `approved` 状态，因为已驳回和已取消的预约不再占用房间。

这个设计的亮点是，时间冲突不是只靠应用层判断，而是由数据库作为最终兜底。即使并发情况下两个请求同时提交，只要它们插入的是同一房

间的重叠时间区间，数据库约束也会拒绝其中一条非法数据。

【代码：跳到 `course-design-final-schema.sql:887` 附近】

这里还可以顺带展示房间、申请人、状态和时间相关索引。它们服务于房间时间轴查询、后台审核筛选和冲突检测。

5:10-6:05 第 7 页：参与人集合检查

【PPT：第 7 页】

预约参与人规则也比较适合体现数据库课程设计。每条预约至少 3 名参与人，并且申请人必须在参与人集合中唯一出现。

【代码：跳到 `course-design-final-schema.sql:903` 附近】

先看 `facility_reservation_participants`。它用 `(reservation_id, user_id)` 作为复合主键，防止同一个用户在同一预约里重复出现。

【代码：跳到 `course-design-final-schema.sql:948` 附近】

再看 `facility_validate_reservation_participants()`。这个规则涉及多行统计，还要比较参与人表和预约主表的 `applicant_id`，普通 `CHECK` 约束做不了，所以这里使用触发器函数统计参与人数量、申请人标记数量和申请人匹配数量。

【代码：跳到 `course-design-final-schema.sql:1037` 附近】

这里的触发器是 `deferrable initially deferred`，也就是延迟到事务提交时检查。原因是创建预约通常先插主表，再插参与人，中间态可能暂时不足 3 人；延迟触发器检查的是事务提交前的最终集合，这更符合实际写入流程。

6:05-6:45 第 8 页：审核状态一致性

【PPT：第 8 页】

预约状态不是一个孤立的枚举字段。比如 `approved` 必须有审核人和审核时间；`rejected` 必须有审核人、审核时间和驳回原因；`cancelled` 必须有取消人和取消时间。

【代码：跳到 `course-design-final-schema.sql:833` 附近】

这里展示 `facility_reservations_status_consistency_chk`。这个 `CHECK` 约束把状态和相关字段绑定起来，防止出现“状态是已通过，但没有审核时间”或者“状态是待审核，但已经有驳回原因”这种脏数据。

这个设计体现的是数据库完整性约束：状态流转不仅在页面上控制，也要在数据库层保证字段组合合法。

6:45-7:30 第 9 页：功能房预约运行验证

【PPT：第 9 页】

这一页展示功能房预约的运行结果。

【网页：打开 `/facilities`】

先看功能房时间轴。合法预约提交后，会在对应房间和时间段上显示占用条，说明预约表、房间表和时间窗口查询能够联动。

如果现场方便，可以尝试选择同一房间的重叠时间段；如果不方便现场操作，就直接说明 PPT 中间截图展示了冲突拦截结果。这个结果对应刚才的重叠预查和数据库排斥约束。

【网页：打开 `/console/facilities/reservations`】

再看后台预约审核页面。待审核、已通过、已驳回这些状态，最终都对应预约表里的状态字段、审核人、审核时间和驳回原因。这里验证了状态一致性约束和审核流程设计。

【回 PPT：第 10 页】

7:30-8:10 第 10 页：课程资源分享整体数据库设计

【PPT：第 10 页】

第三个模块是课程资源分享。这个模块的数据库重点是教学层级、资源审核状态、资源去重、下载事实、积分事件和榜单统计。

整体上，`majors`、`courses` 和 `major_leads` 表达专业、课程和专业负责人的关系；`course_resources` 是资源主体表；下载和积分没有只放在汇总字段里，而是拆成 `course_resource_download_events` 和 `course_resource_score_events` 这样的事件事实表。

这样设计的好处是，资源主体记录当前状态，事件表记录发生过的事实，排行榜和统计结果都有可追溯来源。

8:10-9:15 第 11 页：受控冗余、互斥字段与去重

【PPT：第 11 页】

课程资源里有一个受控冗余设计：资源表同时保存 `course_id` 和 `major_id`。严格来说课程可以决定专业，所以 `major_id` 是冗余字段；但它能让专业范围过滤、专业负责人审核、排行榜统计更方便。

【代码：跳到 `course-design-final-schema.sql:1177` 附近】

这里展示 `course_resources_course_major_fk`。它用 `(course_id, major_id)` 复合外键引用课程表，保证资源保存的专业必须等于课程真实所属专业。也就是说，这个冗余字段不是随意冗余，而是被数据库约束控制的冗余。

【代码：跳到 `course-design-final-schema.sql:1185` 附近】

再看 `course_resources_file_or_link_chk`。资源分为文件型和外链型，文件型不能混入外链字段，外链型不能混入文件字段；非草稿状态还必须补全对应资源明细。这体现了字段互斥和状态相关完整性约束。

【代码：跳到 `course-design-final-schema.sql:1287` 附近】

最后看两个去重索引：文件资源按同一课程下的 `sha256` 去重，外链资源按规范化 URL 去重。它们都是带条件的唯一索引，只约束 `deleted_at is null` 的有效资源，所以软删除后的历史资源不会影响重新提交。

9:15-10:15 第 12 页：审核流程、积分事件与统计事实

【PPT：第 12 页】

资源也有完整状态流转：草稿、待审核、已发布、已驳回、已下架。和功能房预约一样，状态不能只是一个枚举值，还要和提交时间、审核时间、发布时间、下架时间保持一致。

【代码：跳到 `course-design-final-schema.sql:1233` 附近】

这里展示 `course_resources_status_consistency_chk`。例如 `pending` 必须有提交时间但不能有审核字段；`published` 必须有审核时间和发布时间；`unpublished` 必须有发布时间和下架时间。

【代码：跳到 `course-design-final-schema.sql:1333` 附近】

再看 `course_resource_score_events`。积分不是直接覆盖到一个数字里，而是记录成事件事实。这里有 `delta > 0` 检查，还有 `(user_id, resource_id, event_type)` 唯一约束，用来保证同一个用户、同一个资源、同一种事件只加一次分。

【代码：跳到 `course-design-final-schema.sql:1347` 附近】

这里还通过复合外键保证积分事件的 `major_id` 与资源专业一致，积分用户也必须对应资源作者。也就是说，统计事件和资源主体之间仍然有数据库层的一致性约束。

【代码：跳到 `course-design-final-schema.sql:1370` 附近】

如果时间允许，可以展示最佳资源相关触发器。最佳推荐单独建表，并且只有已发布资源才能成为最佳；资源离开发布态后，触发器会撤销最佳关系。

10:15-11:15 第 13 页：课程资源运行验证与总结

【PPT：第 13 页】

这一页是课程资源的运行验证。

【网页：打开 `/console/resources` 或资源详情页】

先看资源发布详情。这里可以看到课程、专业、类型、审核时间、发布时间、作者和下载次数。它验证了 `course_resources` 中资源主体、审核状态、发布时间和统计冗余字段能够正常回显。

【网页：打开 `/resources/leaderboard`】

再看积分与下载榜。下载榜来自下载事件和下载次数统计，积分榜来自积分事件聚合。这里验证了刚才说的事件事实表设计：统计结果不是凭空来的，而是可以追溯到下载事件和积分事件。

【回 PPT：第 13 页】

最后总结一下，本次课程设计围绕三个数据库子模块展开：平台基础体现了一对一扩展、多对多授权、部门闭包表和审计只追加；功能房预约体现了时间区间排斥约束、参与人延迟触发器、状态一致性和并发兜底；课程资源体现了受控冗

余、字段互斥、条件唯一索引、审核状态约束和事件事实表。

整体来说，这个项目不是简单的页面增删改查，而是从需求中的业务规则出发，把 E-R 建模、关系模式、完整性约束、索引、触发器、事务一致性和运行验证落实到了一个可运行系统中。我的讲解到这里结束，谢谢老师观看。