一、课程回顾与问题解答 (答案见yk的PPT，这一部分是语音转文字+AI回答)

1. **函数依赖与BCNF分解示例 (学生-运动会-学院)**
    - **背景:** 关系 R(学号, 项目名称, 名次, 学院)
    - **问题1: 给出函数依赖**
        - 根据“每个学生可参加多个运动会，每个项目得一个名次” -> (学号, 项目名称) → 名次
        - 根据“每个学生只属于一个学院” -> 学号 → 学院
    - **问题2: 说明为何不是BCNF**
        - **BCNF判断标准:** 对于关系R上的每个非平凡函数依赖 X → Y，X 都必须是 R 的超码。
        - **分析:**
            - 候选码是 {学号, 项目名称}。
            - 对于函数依赖 学号 → 学院:
                - 它是一个非平凡函数依赖 (学院 不是 学号 的子集)。
                - 决定因素 X = {学号}。
                - {学号} 不是关系 R(学号, 项目名称, 名次, 学院) 的超码 (因为它不能唯一确定项目名称和名次)。
            - 因此，关系R不满足BCNF。
    - **问题3: 将关系分解为BCNF，并判断分解是否具有无损连接性和依赖保持性**
        - **分解:** 针对违反对BCNF的函数依赖 学号 → 学院 进行分解：
            - R1(<u>学号</u>, 学院) (函数依赖: 学号 → 学院)
            - R2(<u>学号, 项目名称</u>, 名次) (函数依赖: (学号, 项目名称) → 名次)
        - **无损连接判断:**
            - R1 ∩ R2 = {学号}
            - {学号} 是 R1 的码。
            - 因此，该分解是无损连接的。 (BCNF分解算法总是能保证无损连接性)
        - **依赖保持判断:**
            - 原始函数依赖集 F = { (学号, 项目名称) → 名次, 学号 → 学院 }
            - 分解后，(学号, 项目名称) → 名次 可以在 R2 中直接判断。
            - 学号 → 学院 可以在 R1 中直接判断。
            - 所有原始函数依赖都被保持了。
    - *强调:* 书写时要清晰定义R1, R2的属性和主码，并列出其上的函数依赖。
2. **关系代数示例 (超市数据库)**
    - 表：
        - 客户(<u>客户编号</u>, 姓名, 年龄, 住址, ...)
        - 商品(<u>商品编号</u>, 名称, 规格, 生产厂家, 单价, ...)
        - 交易(<u>交易编号</u>, 客户编号, 商品编号, 购买数量, 交易日期, ...)
    - **查询1: 购买了单价小于100的商品的客户姓名和年龄。**
        - $\pi_{姓名,年龄}$ (($\sigma_{单价<100}$ (商品)) ⋈ 交易 ⋈ 客户)
        - *步骤解释*
            1. $\sigma_{单价<100}$ (商品): 从“商品”表中选择出单价小于100的商品记录。
            2. (...) ⋈ 交易: 将上一步结果与“交易”表根据共同属性“商品编号”进行自然连接。
            3. (...) ⋈ 客户: 将上一步结果与“客户”表根据共同属性“客户编号”进行自然连接。
            4. $\pi_{姓名,年龄}$ (...): 从最终连接结果中投影出“姓名”和“年龄”属性。
    - **查询2: 购买了生产厂家是“华为”的商品的客户姓名和商品名称、规格。**

- $\pi_{姓名,名称,规格}$ (客户 ⋈ 交易 ⋈ ($\sigma_{生产厂家='华为'}$ (商品)))

3. **SQL实验题回顾与易错点** (如`EXISTS`/`NOT IN`，子查询别名，`NULL`值处理，聚合函数与`GROUP BY`，复杂连接查询等，具体内容参照之前的笔记)
4. **体检系统数据库设计概要** (作为E-R图和关系模式转换的实例背景，涉及实体如“体检人员”、“医生”、“科室”、“检查项目”、“体检报告”等，以及它们之间的联系，具体设计参照之前的笔记)

二、期末复习总览

1. **考试结构**
   - 选择题
   - 填空题
   - 大题 (关系代数、SQL语句、DB设计、规范化)
2. **各章节重点**
   - **第一章: 绪论**
     - **数据库系统 vs 文件系统:** 数据库系统的优点 (数据共享性高、冗余度低、易扩充、数据独立性高、数据完整性、安全性、并发控制、故障恢复、支持标准化等)。
     - **数据抽象的三个层次 (三级模式结构):**
       - **物理层 (内模式 / 物理模式):** 描述数据在存储介质上的组织方式，是数据在数据库内部的实际存储表示。对应物理数据独立性。
       - **逻辑层 (概念模式 / 模式):** 描述数据库中全体数据的逻辑结构和特征，是所有用户公共的数据视图。是数据库系统模式结构的中间层。对应逻辑数据独立性。
       - **视图层 (外模式 / 用户模式 / 子模式):** 描述数据库用户能够看见和使用的局部数据的逻辑结构和特征，是数据库用户的数据视图。一个概念模式可以有多个外模式。
     - **二级映像功能与数据独立性:**
       - **外模式/概念模式映像:** 保证逻辑数据独立性 (当概念模式改变时，只要不影响外模式，应用程序不变)。
       - **概念模式/内模式映像:** 保证物理数据独立性 (当内模式改变时，概念模式和应用程序不变)。
     - **数据库系统组成:** 数据库(DB)、数据库管理系统(DBMS)、数据库管理员(DBA)、应用程序、用户。
     - **索引 (Index) 概述:**
       - 定义: 索引是数据库中为了提高查询效率而创建的一种辅助性数据结构。
       - 作用: 主要作用是**加快数据的检索速度。**
       - 代价: 占用额外的存储空间；进行增、删、改操作时需要维护索引，**导致这些操作速度略慢。**
   - **第二、六章: 关系模型与关系代数**
     - **关系模型基本概念:** 关系、元组、属性、域、候选码、主码、外码。
     - **关系代数的五个基本运算:**
       - 选择 (σ): $\sigma_F(R)$ - 从关系R中选择满足条件F的元组。
       - 投影 (Π): $\Pi_A(R)$ - 从关系R中选择出若干属性列A组成新的关系 (并去除重复元组)。
       - 并 (∪): R ∪ S - R和S的元组合并 (要求R和S具有相同的目，且相应属性的域相同)。
       - 差 (-): R - S - 从R中去掉与S相同的元组。
       - 笛卡尔积 (×): R × S - R中的每个元组与S中的每个元组进行串接。
     - **扩展/专门运算:**
       - 交 (∩): R ∩ S = R - (R - S) 或 S - (S - R)。
       - 连接 (⋈):

        - θ连接: $R \bowtie_{A\theta B} S = \sigma_{A\theta B}(R \times S)$
        - 等值连接: θ为 '=' 的连接。
        - 自然连接:特殊的等值连接，要求比较的分量是同名属性组，并在结果中去掉重复的同名属性列。
      - 除 (÷): R ÷ S，用于求解"至少/所有"这类查询。
    - 能够用关系代数表达式描述查询需求。
  - **第三、四、五章: SQL语言** (yk实验二、三、四、五、七)
    - **DDL (Data Definition Language):**
      - `CREATE TABLE`: 定义表结构 (列名、数据类型、约束)。
      - `ALTER TABLE`: 修改表结构 (添加/删除列、修改列定义、添加/删除约束)。
      - `DROP TABLE`: 删除表。
      - `CREATE VIEW`: 创建视图。
      - `DROP VIEW`: 删除视图。
      - `CREATE INDEX`: 创建索引。
      - `DROP INDEX`: 删除索引。
    - **DML (Data Manipulation Language):**
      - `INSERT INTO ... VALUES ...`: 插入数据。
      - `UPDATE ... SET ... WHERE ...`: 修改数据。
      - `DELETE FROM ... WHERE ...`: 删除数据。
    - `SELECT` **查询 (核心中的核心):**
      - **基本结构:** `SELECT [DISTINCT] <目标列表达式> FROM <表名或视图名> [WHERE <条件表达式>] [GROUP BY <列名> [HAVING <条件表达式>]] [ORDER BY <列名> [ASC|DESC]]`
      - **连接查询:** `INNER JOIN`, `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, `FULL OUTER JOIN`, `NATURAL JOIN`。
      - **子查询 (嵌套查询):**
        - 不相关子查询 (子查询可以独立执行)。
        - 相关子查询 (子查询的执行依赖于外层查询)。
        - 常用在 `WHERE` 子句中 (与 `IN`, `NOT IN`, `EXISTS`, `NOT EXISTS`, 比较运算符结合)。
        - 常用在 `FROM` 子句中 (作为派生表)。
        - 常用在 `SELECT` 列表后 (作为标量子查询)。
      - **集合操作:** `UNION` (并集，去重), `UNION ALL` (并集，不去重), `INTERSECT` (交集), `EXCEPT` / `MINUS` (差集)。
      - **聚合函数:** `COUNT()`, `SUM()`, `AVG()`, `MAX()`, `MIN()`。
        - `COUNT(*)`: 统计元组个数。
        - 其他聚合函数在计算前会**忽略NULL值**。
        - 通常与 `GROUP BY` 子句一起使用。
      - `CASE WHEN ... THEN ... ELSE ... END` **表达式:** 用于条件逻辑。
    - **视图 (View):**
      - 定义: 视图是从一个或几个基本表（或视图）导出的表，它是一个**虚拟表**，不实际存储数据 (除非是物化视图)，其内容由查询定义。
      - **根本作用: 对用户隐藏特定数据/数据安全**。通过视图，可以限制用户访问数据的范围和类型，保护基表中的敏感信息或复杂结构。
      - **其他作用:**

        - 简化复杂查询：将复杂的查询逻辑封装在视图中，用户只需查询这个简单的视图。
        - 提供一定程度的逻辑数据独立性：当基表的结构发生某些变化时，如果这些变化不影响视图的定义，那么基于该视图的应用程序可以保持不变。
      - **注意点:**
        - 视图的定义存储在数据字典中。
        - 对视图的DML操作可能受限 (例如，基于多表连接、包含聚合函数、`GROUP BY`、`DISTINCT`等的视图通常不可更新)。
    - `NULL` **值:**
      - 表示"未知"或"不适用"的值。
      - `NULL` 与任何值的算术运算结果为 `NULL`。
      - `NULL` 与任何值的比较运算结果为 `UNKNOWN` (三值逻辑: `TRUE`, `FALSE`, `UNKNOWN`)。
      - 判断是否为 `NULL` 必须用 `IS NULL` 或 `IS NOT NULL`。
    - **权限控制 (DCL - Data Control Language):**
      - `GRANT <权限列表> ON <对象类型> <对象名> TO <用户或角色> [WITH GRANT OPTION];`
      - `REVOKE <权限列表> ON <对象类型> <对象名> FROM <用户或角色>;`
      - 权限包括 `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `REFERENCES`, `CREATE TABLE` 等。
  - **第七章: 数据库设计 (E-R模型)**
    - **E-R (Entity-Relationship) 模型元素:**
      - **实体 (Entity):** 客观存在并可相互区分的事物 (如学生、课程)。
      - **属性 (Attribute):** 实体所具有的某一特性 (如学生的学号、姓名)。
      - **联系 (Relationship):** 实体之间的关联 (如学生"选修"课程)。联系的类型: 1:1, 1:N, M:N。
    - **将 E-R 图转换为关系模式的规则:**
      - 一个实体类型转换为一个关系模式，实体的属性即为关系的属性，实体标识符即为关系的主码。
      - 一个M:N联系转换为一个独立的关系模式，其属性为参与联系的各实体的主码 (作为外键联合构成主码) 以及联系自身的属性。
      - 一个1:N联系可以独立转换为一个关系模式，也可以在N端实体对应的关系模式中加入1端实体的主码作为外键。
      - 一个1:1联系可以独立转换为一个关系模式，也可以在任意一端实体对应的关系模式中加入另一端实体的主码作为外键 (通常选择参与度为"部分"的一端，或根据语义)。
    - **完整性约束:**
      - **实体完整性:** 要求关系的主码值不能为 `NULL`，且必须唯一。通过 `PRIMARY KEY` 定义。
      - **参照完整性:** 如果关系R1的外码F_k与关系R2的主码P_k相对应，则R1中任意元组的F_k值，要么等于R2中某个元组的P_k值，要么为 `NULL` (如果F_k允许为 `NULL`)。通过 `FOREIGN KEY ... REFERENCES ...` 定义。
      - **用户定义完整性:** 针对某一具体应用的数据必须满足的语义要求。
        - `NOT NULL`: 约束列值不能为空。
        - `UNIQUE`: 约束列值唯一 (通常允许一个 `NULL`，具体行为看DBMS)。
        - `CHECK (<条件表达式>)`: 约束列值必须满足特定条件。
        - `DEFAULT <值>`: 为列设置默认值。
  - **第八章: 关系规范化理论** (重点和难点)

- **函数依赖 (Functional Dependency, FD):**
  - 定义: 若关系R(U)中，对于属性子集X, Y ⊆ U，如果对于R中任意两个元组t_1, t_2，若$t_1[X] = t_2[X]$，则有$t_1[Y] = t_2[Y]$，则称Y函数依赖于X，或X函数决定Y，记作 X → Y。
  - 平凡FD (Y ⊆ X) vs 非平凡FD。
  - 完全FD vs 部分FD (针对候选码)。
  - 传递FD (X → Y, Y → Z, Y 不函数决定 X, Z 不属于 Y，则 X → Z 是传递依赖)。
- **码 (Key):**
  - **超码 (Superkey):** 能唯一标识元组的属性集。
  - **候选码 (Candidate Key):** 不含多余属性的超码 (即其任何真子集都不是超码)。
  - **主码 (Primary Key):** 从候选码中选定的一个。
  - 主属性: 包含在任何一个候选码中的属性。 非主属性: 不包含在任何候选码中的属性。
- **属性闭包 (X^+):**
  - 定义: 在关系模式R<U, F>中，被属性集X函数决定的所有属性的集合称为X关于F的闭包，记为X^+_F。
  - **计算方法:** 基于Armstrong公理 (自反律、增广律、传递律) 进行推导。
    1. 令 $X^+ = X$。
    2. 重复检查F中的每个FD Y→Z：如果$Y \subseteq X^+$ 且 $Z \not\subset X^+$，则将Z加入X^+。
    3. 直到X^+不再增大为止。
  - **应用:** 判断X是否为超码 (若X^+=U)、判断候选码、判断FD是否被F蕴含 (若$Y \subseteq X^+$，则X→Y被F蕴含)。
- **范式 (Normal Forms, NF):**
  - **第一范式 (1NF):** 关系中的每个属性都是不可再分的原子值。 (这是关系模型的基本要求)
  - **第二范式 (2NF):** 在1NF基础上，消除所有非主属性对任何候选码的部分函数依赖。
  - **第三范式 (3NF):** 在2NF基础上，消除所有非主属性对任何候选码的传递函数依赖。(简洁定义：若X→A，A为非主属性，则X必为超码)
  - **BCNF (Boyce-Codd Normal Form):** 在3NF基础上，消除所有主属性对不包含它的码的部分和传递依赖。(简洁定义：对于每个非平凡函数依赖 X→A，X都必须是关系的超码)。
  - 范式级别越高，数据冗余越小，更新异常越少，但查询时可能需要更多连接。
- **分解 (Decomposition):** 将一个关系模式分解为若干个关系模式。
  - **无损连接分解:** 保证分解后的关系能够通过自然连接恢复原始关系的所有信息，不丢失信息也不产生额外信息。
  - **依赖保持分解:** 保证原始关系中的所有函数依赖在分解后的关系中仍然能够被推导出来 (或者说，F中每个FD的函数闭包都能在某个分解后的R_i上计算出来)。
  - **BCNF分解算法:**
    1. 目标是达到BCNF并保持无损连接 (不一定保持依赖)。
    2. 对于关系模式R，若存在X→Y违反BCNF (X不是超码)，则将R分解为 R1(X Y) 和 R2(R - Y)。
    3. 递归地对R1和R2进行分解，直到所有关系模式都达到BCNF。
  - **3NF分解算法:**
    1. 目标是达到3NF并保持无损连接和依赖。
    2. 找出F的最小依赖集F_{min}。

3. 对于F_{min}中的每个FD X→Y，创建一个关系模式$R_i(XY)$。
4. 如果所有候选码都不在任何一个R_i中，则将任一候选码K作为一个独立的关系模式$R_K(K)$加入。

- **事务管理、并发控制与恢复 (最后部分)**
  - **事务 (Transaction):** 用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的逻辑工作单元。
  - **ACID特性:**
    - **原子性 (Atomicity):** 事务是原子的，其包含的操作要么全部成功执行，要么全部不执行 (回滚)。
    - **一致性 (Consistency):** 事务执行前后，数据库从一个一致性状态转变到另一个一致性状态。如果事务独立执行，它将保持数据库的一致性。
    - **隔离性 (Isolation):** 多个事务并发执行时，一个事务的执行不应被其他事务干扰，即事务之间是相互隔离的，使得每个事务感觉不到其他事务在并发执行。
    - **持久性 (Durability):** 一旦事务成功提交，其对数据库中数据的改变就是永久性的，即使系统发生故障也不会丢失。
  - **事务状态:** 活动(Active)、部分提交(Partially Committed)、失败(Failed)、中止(Aborted)、提交(Committed)。 (关键操作：`COMMIT` 确认提交, `ROLLBACK` / `ABORT` 撤销中止)
  - **并发控制:** 协调多个并发事务的执行，以保证数据库的一致性并避免相互干扰。
    - 并发可能导致的问题: 丢失修改、读脏数据、不可重复读、幻读。
    - **可串行化 (Serializability):** 并发事务的执行结果与按某一顺序串行执行它们的结果相同。这是衡量并发控制正确性的标准。
    - 冲突可串行化 (Conflict Serializability): 通过交换不冲突操作的顺序，可以将一个并发调度转换为某个串行调度。
  - **封锁协议 (Locking):** 实现并发控制的主要技术。
    - 锁的类型: 共享锁 (S锁/读锁)、排他锁 (X锁/写锁)。
    - **两段锁协议 (2PL / Two-Phase Locking):**
      - **生长阶段 (Growing Phase):** 事务可以获得锁，但不能释放任何锁。
      - **衰退阶段 (Shrinking Phase):** 事务可以释放锁，但不能再获得任何新的锁。
      - 两段锁协议是保证冲突可串行化的。但可能导致死锁。
      - 变种: 严格两段锁 (事务必须持有所有X锁直到提交)、强两段锁 (事务必须持有所有锁直到提交)。
  - **死锁 (Deadlock):** 两个或多个事务互相等待对方释放资源而都无法继续执行的现象。
    - 处理方法：死锁预防 (破坏死锁产生的四个必要条件之一)、死锁检测与解除 (允许死锁发生，定期检测，然后选择一个或多个事务回滚以解除死锁)。