数据库期末复习

考题

5道选择 10‘

3道关系代数 15’

5道sql 25‘

三道函数依赖理论、范式 15’

事务管理：冲突可串行、2PL、恢复算法 15‘

ER设计 20’

补充一下查询树

FROM CSDN：期末建议

1. 在总复习前请**先刷2018年的那套题目**，这套题目的题型非常经典，可以通过做这套题目来明确期末考试题型，每题考什么都是固定的，这套题目的答案和解析是我很认真做的，里面给出了每种题型的复习方法和算法，如果刷完这套题并且掌握了精解精析的内容，直接上考场应该也能考上85了。切记，**精解里面的算法很重要**，务必掌握！
2. 复习完2018的卷子后，**开始背诵《往年简答题汇总》**，不算很多，一天就可以背完，虽然不多但是都是高频内容，每年重复率很高，甚至有全部都是原题的情况，这个可以帮助你在考场上快速完成简答部分，进入后续思维量较多的部分。（这里面的答案都是精精精精精校过的，可以确保万无一失，直接背诵就可以了）背完以后，后面到了刷真题阶段的时候就可以在做简答部分的时候带着复习背诵内容了。
3. 做完1和2两项，下面进入比较痛苦的复习阶段了，**对着《db总结》复习课堂内容**，只要能把该ppt上提及的内容复习到就可以，没有提及到的是不会考的（关于这点要相信任课老师的复习ppt）复习完一遍后，最好能复盘一遍，大概需要花费4天。
4. 恭喜你，现在你可以开始刷题了，按**从新题往老题的方向刷题（2020-2007）**，零几年的题如果实在来不及可以只做选择题。刷题的时候记得复盘之前背过的简答题。
5. 最后，**考前必看**。上考场前可以浏览一遍《数据库复习要点》，这个不需要花费太多时间，1-2小时即可。（也可以直接拿《db总结》ppt快速浏览一遍）。

- 说在最后：
  这份材料是我精心整理的一份材料，就是靠着这份材料我《数据库系统原理》最后取得了很高的分数。这份材料不能保证你上100分，但是只要掌握了其中应该掌握的内容，考上95+还是很容易的。

考纲

一些概念

- 表：关系
- 元祖：行
- 属性：列
- 关系模式：类型定义
- 关系实例：值
- 域：属性的取值范围
- 原子域：域中元素被认为是不可再分的单元
- 空值：特殊的值，标明未知或不存在
- 码：
  - 超键是一个或多个属性的集合，将这些属性组合在起可以唯一地标识出一个元组
  - 超码的任意超集也是超码。最小的超码是候选码。
  - 被设计者选定的候选码是主键。
  - 主键约束
- r1 关系的A属性（集）到 r2 关系的主键B的外键约束 (foreign-key constraint)
  - 引用关系：r1
  - 被引用关系：r2
- 引用完整性约束：引用完整性约束 (referential integrity constraint) 要求引用关系中的任意元组在指定属性出现的取值也必然出现在被引用关系中至少1个元组的指定属性上
- 模式图：数据库梶式的图形化表示，它显示了数据库的关系、关系的屈性以及主码和外码。
- 查询语言类型
  - 命令式：
  - 函数式：计算被表示为对函数的求值
  - 声明式：只需要描述所需要的信息，不用给出该信息的具体步骤序列或函数调用

1. 关系代数

2. SQL

DDL

- Create/ Drop/Alter table;
- Create/ Drop/Alter view;
- Create/ Drop index;

代码块

```
-- 创建表格
CREATE TABLE Employee (
    emp_ID INT PRIMARY KEY,
    emp_name VARCHAR(50) NOT NULL,
    phoneNum VARCHAR(20) UNIQUE,
    salary DECIMAL(10,2),
    department_id INT,
    manager_id INT,
    birth_date DATE,
    FOREIGN KEY (department_id) REFERENCES department(department_id),
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
)
-- 创建联系集
CREATE TABLE employee_projects (
    employee_id INT,
    project_id INT,
    hours_worked DECIMAL(5,2),
    PRIMARY KEY (employee_id, project_id),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
    FOREIGN KEY (project_id) REFERENCES projects(project_id));

-- 创建视图
CREATE VIEW EmployeeOA AS
    SELECT name, position
    FROM Employee

-- 创建部门摘要视图
CREATE VIEW department_summary AS
SELECT
    d.department_id,
    d.department_name,
    COUNT(e.employee_id) AS employee_count,
    AVG(e.salary) AS avg_salary
FROM
    departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name;
```

代码块

```sql
--增加列
ALTER TABLE Employee
ADD email VARCHAR(50);

--修改
ALTER TABLE Employee
MODIFY department_name VARCHAR(30);

--ALTER添加检查约束
ALTER TABLE employees
ADD CONSTRAINT salary_check CHECK (salary > 0);
--添加外键约束
ALTER TABLE projects
ADD CONSTRAINT fk_project_manager
FOREIGN KEY (manager_id) REFERENCES employees(employee_id);

--ALTER重命名
ALTER TABLE employees
RENAME COLUMN phone_number TO contact_number;
```

代码块

```sql
DROP TABLE employee_projects;
```

DML-query_BASIC

代码块

```sql
SELECT *
FROM Employee e JOIN department d ON d.department_id = e.department_id
WHERE d.department_name = 'SE'
AND e.salary BETWEEN 5000 AND 10000
AND e.email LIKE '%@company.com';
AND e.birth_date > '2023-01-01'
ORDER BY salary DESC, e.emp_name ASC;

SELECT DISTINCT employ_id, 12*salary AS annual_salary
FROM employee;

SELECT e.employee_name, e.salary, e.email
FROM Employee e JOIN department d ON e.department_id = d.department_id
```

DML-query_Set

代码块

```
SELECT employee_id, first_name FROM employees WHERE department_id = 10
UNION/INTERSECT/EXCEPT (ALL)
SELECT employee_id, first_name FROM employees WHERE salary > 8000;
```

DML-query_aggregation

Basic Aggregation

代码块

```
SELECT
    COUNT(*) AS total_employee,
    AVG(salary) AS avg_salary,
    SUM(salary) AS total_salary,
    MAX(salary) AS max_salary,
    MIN(salary) AS min_salary
FROM employee
```

带group-having的Aggregation

代码块

```
SELECT department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id --GROUPBY后面的属性一定要在SELECT里
HAVING avg_salary > 8000;  -- HAVING过滤分组后的结果

--多列分组聚合
SELECT
    department_id,
    job_title,
    COUNT(*) AS count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id, job_title;
```

带NULL的Aggregation

除了count(*) , 其它聚集函数都忽略null

DML-query_Nested Subqueries

代码块

```sql
-- 查询参与了项目/一个项目都没参与的员工
SELECT * FROM employee
WHERE employee_id IN/NOT IN (
    SELECT DISTINCT employee_id FROM employee_projects --参与了至少一个项目的员工
)

-- 工资高于IT部门任一/所有员工的员工
SELECT * FROM employee
where salary > SOME/ALL(
    SELECT salary FROM employee
    WHERE department_id = (
        SELECT department_id FROM department
        WHERE department_name = 'IT'
    )
)

--找出至少有一项记录/没有任何项目记录的员工
--当子查询结果为空时，NOT EXISTS 条件成立
SELECT * FROM employees e
WHERE EXISTS/NOT EXISTS (
    SELECT 1 FROM employee_projects
    WHERE employee_id = e.employee_id
);

-- 查询项目参与情况
WITH project_stats AS (
    SELECT
        project_id,
        COUNT(*) AS participant_count,
        SUM(hours_worked) AS total_hours
    FROM employee_projects
    GROUP BY project_id
)
SELECT p.project_name, ps.*
FROM projects p JOIN project_stats ps ON p.project_id = ps.project_id;

```

JOIN操作

代码块

```sql
SELECT * FROM departments NATURAL JOIN employees;

SELECT * FROM employees JOIN departments USING(department_id);

SELECT e.*, d.department_name
FROM employees e LEFT OUTER JOIN departments d ON e.department_id =
d.department_id;
```

标量子查询

代码块

```sql
-- 查询员工信息及其部门人数
SELECT
    e.*,
    (SELECT COUNT(*) FROM employee
     WHERE department_id = e.department_id) AS dept_employee_count
FROM employees e;

-- 查询项目及最早参与员工
SELECT
    p.*,
    (SELECT MIN(join_date) FROM employee_projects
     WHERE project_id = p.project_id) AS first_join_date
FROM projects p;
```

3. 查询优化

4. E-R图

4.1 给关系模式画ER图

冗余：如果对千每个“上课”实体都存储 课程标识 和 课程名称，名称 存储就是冗余的

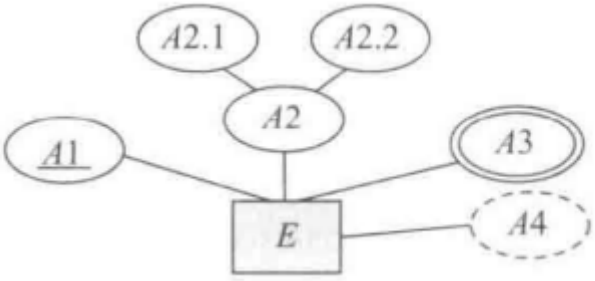
复合属性：可以被划分为子部分，可以有层次。比如address-*street*,city,state,postal _code，street 可以进 步分为 street number stree nam apartment_number

单值属性：对某个特定学生 student lD 只对应1个值。

多值属性：instructor的 phone_num 属性，每位教师可以有零个1个或多个电话号，不同的教师可以有不同数量的电话号——ER图中表示为{phone_num}

派生属性：instructor 实体的 age ，可以从 dateofbirth 计算出来；students_advised可以从与instructor有关联的student实体数计算出来——ER图中表示为age()

实体集E包含
简单属性A1、
复合属性A2、
多值属性A3、
派生属性A4、
以及主码A1



一对一：一个老师最多指导一个学生，一个学生最多找一个老师

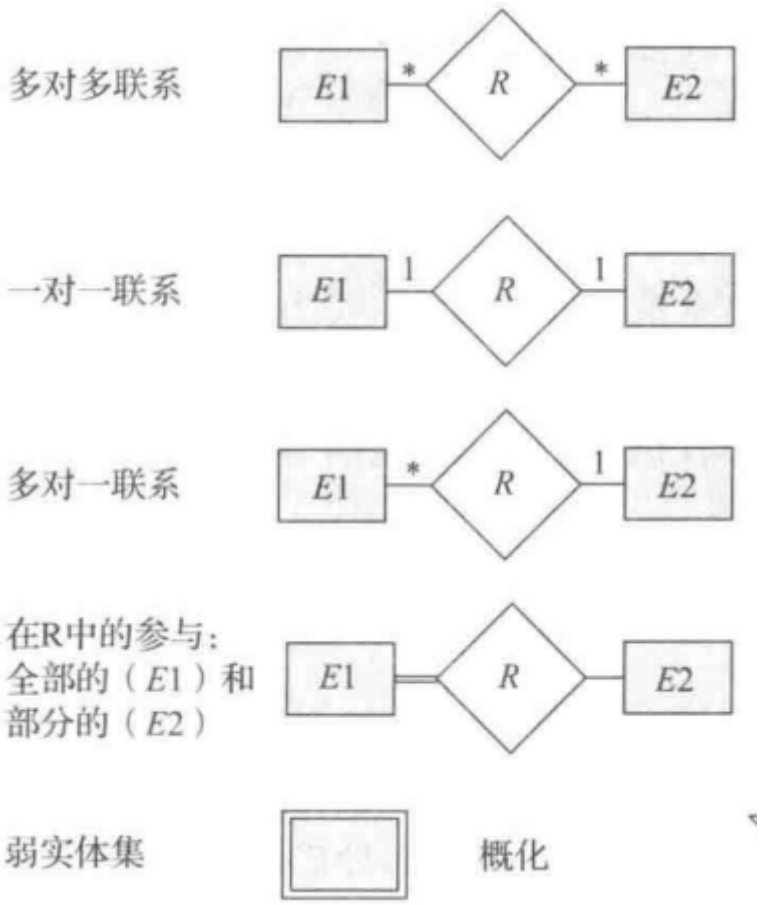
一对多：一个老师可以指导多个学生，一个学生最多找一个老师

多对一：一个老师最多指导一个学生，一个学生可以找多个老师

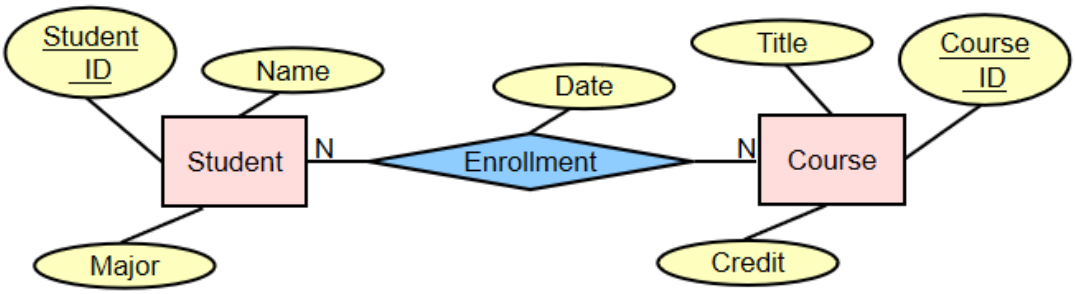
多对多：一个老师可以指导多个学生，一个学生可以找多个老师

每个实体都必须参与至少一个联系，则是**全部参与**（用双线表示），否则是**部分参与**。

比如：每名学生至1位导师，导师可以不收学生，因此学生是全部，导师是部分



N对N的联系集需要单独**建表**



Student_ID	Name	Major

Course_ID	Title	Credit

Student_ID	Course_ID	Date

弱实体集是**不能靠自己唯一标识的实体集**，依靠另一个“强实体集”来确定身份。比如，“床位”这个实体不能单独靠“床号”来区分，只能依赖“宿舍号 + 床号”一起，才能唯一确定一个床位（比如：“3号宿舍的1号床”），“床位”就是一个**弱实体集**，而“宿舍”是它依赖的“强实体集”或“主实体集”。

4.2 根据ER图写对应的关系模型

5. 数据库设计

范式判断

	违反规则	定义	违反的例子
2NF	存在某个**非主属性**，它只依赖于这个组合候选码的**一部分** 左边是码的一部分，且右侧不是主属性	所有**非主属性**都**完全函数依赖**于 R 的每一个候选码	候选码AB，非主属性CD AB->C B->D
3NF	存在**非主属性传递依赖于候选码**时 存在非主属性依赖于不是候选键时	R 中没有**非主属性**传递函数依赖于 R 的任何候选码 满足两个条件之一： a) `X` 左边是一个超码。 b) `Y` 中所有属性都是主属性。	A是候选码，A->B A->C C->D，导致A->C->D，D传递依赖于A

	左侧不是候选键，且右侧不是主属性		
BCNF	函数依赖 `X → Y`，`X` 不是超码 左侧不是候选键	每个非平凡的函数依赖 `X → Y`，`X` 都必须是 R 的一个**超码**	AB是候选码。AB -> CD，D->A，D不是超码
4NF	存在 `X →→ Y`，`X` **不是超码**时	每个非平凡的**多值依赖** `X →→ Y`，`X` 都必须是 R 的一个**超码**。	`课程 →→ 授课老师`，`课程 →→ 参考书`，课程不是超码

- （注：码属性是包含在“候选码”中的属性，也称为“主属性”）

求F+和BCNF分解

💁‍♀️ 画表格，求出候选码

找到违反BCNF的函数依赖，不断分解直到不存在

Fc和3NF分解

💁‍♀️ 画表格，求出候选码

求Fc：右化单，左去冗，去推导，合并

写R：左右合并，不包含候选码则新增一个R为候选码的左右合并

6. 事务管理

6.1 ACID

cr:

事务的ACID特性：事务的ACID特性，需要程序员，并发控制机制和恢复机制共同来保证。

- 原子性：每个事务中的操作要不全作要不全不做
- 一致性: 每个事务的内部操作的逻辑正确，即在事务的开始和结束时，数据库中的数据都是一致的，符合业务逻辑的。
- 隔离性: 当多个事务并发执行时，每个事务都感觉不到别的事务在执行，不受其它事务的影响
- 持久性: 每个事务对数据库的更新都会永久的反映到数据库中保存下来，不会受故障的影响。

属性（Property）	描述（Description）
原子性（Atomicity）	一个事务是不可分割的单位：要么所有操作都执行，要么都不执行。
一致性（Consistency）	一个事务将数据库从一个一致的状态转换到另一个一致的状态。
隔离性（Isolation）	并发执行的事务之间不会互相干扰，中间状态对其他事务是不可见的。
持久性（Durability）	一旦事务提交，其对数据库的修改就是永久性的，即使系统崩溃也不会丢失。

6.2 冲突可串行 conflict serializable

可串行化和可恢复调度，保证事务隔离性和原子性

a. **题一：是否是冲突串行化**的调度，若是，请给出**等价的串行化调度**，若不是，请给出理由——画图，是否带环

b. **题二：**判定该调度**是否是可恢复**的调度——T1 W(A)后T2R(A)，则T2commit必须在T1commit后面

c. **是否无极联**的调度。——T2 读了 T1 的未提交数据，T1T2级联

6.3 两阶段锁2PL

一种并发控制机制，保证并发事务的调度是可串行化的，可恢复的（最好是无极联的）

a. 给定一组事务，按两阶段封锁协议**2PL进行加锁和解锁。**

b. 判定加2PL锁后，**是否可能死锁。**

6.4 恢复算法

保证事务的原子性和持久性

a. 啥时候redo/undo——commit了就redo，没有就undo
b. 值是什么
c. log怎么写

By yzx 2023级