CS213 数据库系统原理 (H) 详细 Lecture Note

基于 Shiqi Yu (SUSTech) 讲义整理

2025 年 12 月 4 日

目录

1 Chapter 1: 数据库系统导论 (Introduction to Databases)

1.1 1.1 数据库系统的本质

数据库系统不仅仅是数据的集合，它是一个负责管理大量、复杂、相关联数据的软件系统。

- **组成要素：**
  - **数据 (Data)：**相关联的数据集合。
  - **程序 (Programs)：**一组用于访问和操作数据的程序。
  - **环境 (Environment)：**一个既方便又高效的使用环境。
- **历史演变：**
  - **早期 (File Systems)：**数据直接存储在文件系统中。存在数据冗余、不一致、访问困难、原子性缺失、并发访问困难及安全问题。
  - **1970 年：**Edgar F. Codd 发表了划时代的论文 *"A Relational Model of Data for Large Shared Data Banks"*，提出了关系模型。
  - **1980s：**SQL 成为工业标准；关系型数据库 (RDBMS) 商业化（如 Oracle, DB2）。
  - **2000s：**大数据时代，NoSQL 系统（如 Google BigTable, Amazon Dynamo）兴起，用于处理非结构化数据和超大规模数据；MapReduce 流行。
  - **2010s：**NewSQL 出现，试图结合 NoSQL 的可扩展性和 SQL 的 ACID 特性；多核内存数据库成为趋势。

1.2 1.2 关系模型 (Relational Model)

- **关系 (Relation)：**即“表”(Table)。
- **元组 (Tuple)：**表中的一行 (Row/Record)，代表一个实体或事实。
- **属性 (Attribute)：**表中的一列 (Column)，代表数据的一个特征。
- **核心思想：**表中的一行数据是相互关联的 (related)，即“关系”一词的由来。
- **集合论基础：**关系是数学上的集合，因此可以对表进行集合运算（选择、投影、连接等）从而产生新的表。

1.3 1.3 键 (Keys)

在关系模型中，**不允许存在重复的行** (No duplicates)。为了区分不同的行，需要使用“键”。

- **超键 (Super Key)：**能唯一标识一个元组的一个或多个属性的集合。
- **候选键 (Candidate Key)：**最小的超键（没有多余属性）。
- **主键 (Primary Key, PK)：**被数据库设计者选中用来作为主要标识符的候选键。通常选择最简单、最稳定的那个。

易错点 / Pitfall

虽然理论上关系集合不允许重复行，但在实际的 SQL 数据库实现中，如果没有显式定义主键或唯一约束，表是可以包含完全重复的行的（这被称为 bag 而不是 set）。这在数据处理中通常是不良设计。

1.4 1.4 范式 (Normalization)

范式化是组织数据以减少冗余和依赖的过程。

- **第一范式 (1NF)**：属性必须是原子的 (Atomic)。
  - 错误示例：一个单元格内存放"Zhang San, Li Si"。
  - 正确做法：一行只存一个名字，或者分拆成多行。
- **第二范式 (2NF)**：满足 1NF，且非主属性必须**完全依赖**于主键（消除部分依赖）。
  - 如果主键是 (StudentId, CourseId)，那么"StudentName" 不应该放在这张表里，因为它只依赖于 StudentId，而不依赖于 CourseId。
- **第三范式 (3NF)**：满足 2NF，且非主属性不依赖于其他非主属性（消除传递依赖）。
  - 例如：(MovieId, DirectorId, DirectorName)。DirectorName 依赖于 DirectorId，而 DirectorId 依赖于 MovieId。这构成了传递依赖，应该拆分为两张表。

2 Chapter 2: SQL 基础 (Introduction to SQL)

2.1 2.1 SQL 语言概览

SQL (Structured Query Language) 起源于 IBM 的 SEQUEL 项目，旨在提供一种类似英语的结构化查询语言。

- **DDL (Data Definition Language)**：定义数据结构。核心指令：`CREATE`, `ALTER`, `DROP`。
- **DML (Data Manipulation Language)**：操作数据内容。核心指令：`INSERT`, `UPDATE`, `DELETE`, `SELECT`。

修正与现代观点 / Modern Practice

SQL 是一种声明式语言 (Declarative)，你告诉数据库**你想要什么 (What)**，而不是**怎么做 (How)**。但在性能调优时，理解底层的“怎么做”（执行计划）至关重要。

2.2 2.2 创建表 (CREATE TABLE)

```sql
CREATE TABLE table_name (
    column1 datatype [constraints],
    column2 datatype [constraints],
    ...
);
```

- **标识符规则：**表名和列名通常不区分大小写（Case-insensitive）。建议使用下划线命名法（snake_case），避免使用空格或保留字。
- **数据类型 (Data Types)：**
  - **文本：**`CHAR(n)` (定长，不足补空格), `VARCHAR(n)` (变长), `CLOB/TEXT` (大文本)。

    易错点 / Pitfall

    Oracle 中 `VARCHAR2` 是推荐类型，空字符串 `''` 在 Oracle 中被视为 `NULL`，这与其他数据库不同。
  - **数字：**`INT`, `FLOAT`, `NUMERIC(p,s)` / `DECIMAL(p,s)` (p 为总位数，s 为小数位)。
  - **日期：**`DATE` (通常含时间，Oracle/MySQL 不同), `TIMESTAMP` (更高精度), `DATETIME` (SQL Server)。

2.3 2.3 约束 (Constraints)

约束是保证数据一致性和正确性的关键机制。

- **NOT NULL：**列值不能为空。
- **PRIMARY KEY：**唯一标识行，隐含 NOT NULL 和 UNIQUE。
- **UNIQUE：**列值必须唯一（但通常允许有多个 NULL，视 DBMS 而定）。
- **CHECK：**自定义逻辑校验（例如 `year > 1900`）。

  修正与现代观点 / Modern Practice

  MySQL 在 8.0.16 版本之前会忽略 CHECK 约束，这在当时是一个主要缺陷。现在的版本已支持。
- **FOREIGN KEY (Referential Integrity)：**确保引用的值在父表中存在。
  - 语法：`FOREIGN KEY (col) REFERENCES other_table(col)`。
  - 作用：防止出现“孤儿数据”。删除父表数据时，如果子表有引用，数据库会报错或级联删除（需配置）。

2.4 2.4 插入数据 (INSERT)

```
INSERT INTO table_name (col1, col2) VALUES (val1, val2);
```

- **字符串引用**：标准 SQL 使用**单引号** `'string'`。
- **转义引号**：在字符串中表示单引号，通常使用双写单引号 `'It''s ok'`。
- **日期输入**：强烈建议使用格式转换函数（如 `TO_DATE` 或 `DATE()`）显式输入日期，不要依赖隐式转换，因为不同地区的日期格式（DD/MM/YYYY vs MM/DD/YYYY）极易混淆。

3 Chapter 3: 单表查询 (Retrieving Data from One Table)

3.1 3.1 SELECT 基础

- `SELECT *`：检索所有列。

> **易错点 / Pitfall**
>
> **程序开发禁忌**：在生产代码中尽量避免使用 `SELECT *`。如果表结构变更（增加列），可能会导致程序崩溃或不仅浪费网络带宽。应显式列出需要的列名。

- **投影 (Projection)**：选择特定的列。
- **选择 (Selection/Restriction)**：使用 `WHERE` 子句筛选行。

3.2 3.2 过滤与运算符 (WHERE Clause)

- **比较符**：`=`, `<>` (或 `!=`), `<`, `>`, `<=`, `>=`。
- **逻辑符**：`AND`, `OR`, `NOT`。
  - **优先级**：`AND` 的优先级高于 `OR`。混合使用时务必使用括号 `()` 明确意图。
- **范围与列表**：`BETWEEN a AND b` (包含边界), `IN (a, b, c)`。
- **模糊匹配**：`LIKE`。
  - `%`：匹配任意长度字符（包括 0 个）。
  - `_`：匹配任意单个字符。
  - 大小写敏感性取决于 DBMS 和 Collation 设置。

3.3 3.3 NULL 的陷阱

易错点 / Pitfall

NULL 不等于 NULL。

- NULL 代表“未知”或“不存在”。
- 任何与 NULL 进行的比较运算（`=`, `!=`, `>`, `<`）结果都是 **Unknown**（在 SQL 中会被视为 False）。
- **错误写法：** `WHERE col = NULL` （永远不返回任何行）。
- **正确写法：** `WHERE col IS NULL` 或 `WHERE col IS NOT NULL`。
- **三值逻辑：** TRUE, FALSE, UNKNOWN。注意 `NOT UNKNOWN` 仍然是 `UNKNOWN`。

3.4 3.4 函数与转换

- **字符串：** `UPPER()`, `LOWER()`, `SUBSTR()`, 连接符 `||` (ANSI) 或 `+` (SQL Server) 或 `CONCAT()`。
- **日期：** 日期运算极其依赖 DBMS。
  - `DATEADD`, `DATEDIFF` (SQL Server)。
  - `date + interval '1' day` (PostgreSQL/MySQL)。
  - 比较 `DATETIME` 和 `DATE` 时要注意时间部分可能导致相等判断失败。
- **类型转换：** `CAST(col AS type)` 是标准语法。隐式转换（如字符串与数字比较）可能导致性能问题（索引失效）。

3.5 3.5 CASE 表达式

用于在 SQL 中实现 if-else 逻辑。

```sql
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ELSE result3
END
```

或者：

```sql
CASE column
    WHEN val1 THEN res1
    ELSE res2
END
```

注意： 如果省略 `ELSE` 且没有匹配项，默认返回 `NULL`。

4 Chapter 4: 聚合与连接 (Aggregation and Joins)

4.1 4.1 去重 (DISTINCT)

- `SELECT DISTINCT col1, col2 ...`：去除结果集中所有指定列组合完全相同的重复行。
- 这会将结果集转化为真正意义上的“关系”（集合）。

4.2 4.2 聚合函数 (Aggregate Functions)

- **常见函数：**`COUNT(*)` (统计所有行), `COUNT(col)` (统计非 NULL 值), `SUM()`, `AVG()`, `MIN()`, `MAX()`。
- **NULL 处理：**聚合函数（除 `COUNT(*)` 外）会自动忽略 NULL 值。
- **GROUP BY：**将数据分组，对每组应用聚合函数。

> **易错点 / Pitfall**
>
> **SELECT 列表限制：**在使用 `GROUP BY` 时，SELECT 子句中只能包含聚合函数或 `GROUP BY` 中列出的列。不能选择非分组列，因为它们的值是不确定的。

- **HAVING：**对**分组后**的结果进行过滤（例如：筛选出平均分大于 60 的班级）。`WHERE` 是在分组前过滤。

4.3 4.3 连接 (Joins)

连接是将多张表的数据结合起来的核心操作。

- **内连接 (INNER JOIN)：**仅返回两张表中满足连接条件的行。
- **语法演变：**

> **修正与现代观点 / Modern Practice**
>
> **显式 JOIN (ANSI-92) vs 隐式 JOIN (ANSI-89)：**讲义中提到了使用 `FROM A, B WHERE A.id = B.id` 的写法。**现代最佳实践：**强烈建议使用 `FROM A JOIN B ON A.id = B.id`。原因：1. 分离了连接逻辑 (ON) 和过滤逻辑 (WHERE)，可读性更高。2. 防止意外遗漏 WHERE 条件导致笛卡尔积 (Cartesian Product)，造成性能灾难。

- **多表连接：**可以链式进行 `JOIN ... ON ... JOIN ... ON ...`。

5 Chapter 5: 高级连接与子查询 (Advanced Joins and Subqueries)

5.1 5.1 外连接 (Outer Joins)

- **LEFT OUTER JOIN：**返回左表的所有行。如果右表没有匹配，右表的列显示为 NULL。

- **RIGHT OUTER JOIN**：返回右表的所有行（不常用，通常习惯改写为 LEFT JOIN）。
- **FULL OUTER JOIN**：返回两张表的所有行，任一边不匹配则补 NULL（MySQL 不直接支持，需用 UNION 模拟）。
- **陷阱**：如果在 `WHERE` 子句中对右表（Left Join 的被连接表）的列加条件，会将外连接“退化”为内连接。
  - 错误：`... LEFT JOIN B ON ... WHERE B.col = 1`
  - 正确：`... LEFT JOIN B ON ... AND B.col = 1`（如果是连接条件）或检查 `B.col IS NULL`。

5.2 5.2 集合操作 (Set Operators)

要求两个查询结果的列数相同，且对应列的数据类型兼容。

- **UNION**：合并结果并**去重**。
- **UNION ALL**：合并结果**不去重**。效率通常比 UNION 高，因为不需要排序去重。如果确定无重复或需要保留重复，优先使用此项。
- **INTERSECT**：交集（两边都有的行）。
- **EXCEPT / MINUS**：差集（左边有但右边没有的行）。

5.3 5.3 子查询 (Subqueries)

- **标量子查询**：返回单一值的子查询，可用在 `SELECT` 列表或 `WHERE` 比较中。
- **IN / NOT IN**：判断值是否在列表中。

> **易错点 / Pitfall**
>
> **NOT IN 与 NULL**：如果子查询结果中包含任何 NULL 值，`NOT IN` 将永远返回空结果（因为 `x <> NULL` 是 Unknown）。在这种情况下应优先使用 `NOT EXISTS`。

- **EXISTS / NOT EXISTS**：检查子查询是否返回行。通常比 `IN` 性能更好，且对 NULL 处理更直观。
- **相关子查询 (Correlated Subquery)**：子查询内部引用了外部查询的列。对每一行外部数据，子查询都要执行一次（但在现代优化器下往往会被优化为 Join）。

6 Chapter 6: 排序与窗口函数 (Ordering and Window Functions)

6.1 6.1 排序 (ORDER BY)

- `ORDER BY col [ASC|DESC]`。

- 多列排序：`ORDER BY col1 ASC, col2 DESC`。
- **NULL 排序**：不同 DBMS 处理不同。Oracle 默认 NULL 最大（排最后），SQL Server 默认 NULL 最小。可使用 `NULLS FIRST/LAST` 语法（如果支持）。
- **分页**：
  - MySQL/PG: `LIMIT x OFFSET y`
  - SQL Server: `TOP x` 或 `OFFSET ... FETCH NEXT ...`
  - Oracle: `ROWNUM` (旧) 或 `FETCH FIRST ...` (12c+)

6.2 6.2 窗口函数 (Window Functions)

用于在不聚合行的情况下计算聚合值（如“每行的累计总和”、“每行所属组的平均值”）。

- 语法：`Function() OVER (PARTITION BY ... ORDER BY ...)`
- **排名函数**：
  - `ROW_NUMBER()`：1, 2, 3, 4（即使值相同也强行排序，无并列）。
  - `RANK()`：1, 2, 2, 4（有并列，排名跳跃）。
  - `DENSE_RANK()`：1, 2, 2, 3（有并列，排名连续）。
- **聚合窗口**：如 `SUM(salary) OVER (PARTITION BY dept)` 计算部门总薪资，附加在每一名员工记录后。

7 Chapter 7: 模糊搜索与事务 (Fuzzy Search and Transactions)

7.1 7.1 模糊搜索

- **问题**：Typos（拼写错误），不同语言的拼写差异（Shenzhen vs Shenzen）。
- **解决方案**：
  - `LIKE`：简单的模式匹配，但无法处理拼写错误，且 `%abc` 无法利用索引。
  - **Soundex**：基于发音的算法（如将姓名转换为发音代码比较）。
  - **全文检索 (Full-text Search)**：分词、倒排索引。
  - **自定义距离**：如 Levenshtein Distance（编辑距离）。

7.2 7.2 事务 (Transactions)

事务是逻辑上的一组操作，要么全做，要么全不做。

- **ACID 属性**：
  - **A (Atomicity)**：原子性，不可分割。

  - **C (Consistency)**：一致性，事务前后数据需满足约束。
  - **I (Isolation)**：隔离性，并发事务互不干扰（存在隔离级别）。
  - **D (Durability)**：持久性，提交后数据不丢失。
- **控制语句**：`BEGIN` (或 `START TRANSACTION`), `COMMIT`, `ROLLBACK`。
- **Auto-commit**：很多数据库默认每条语句就是一个事务，需显式关闭或显式开启事务块。

7.3 7.3 数据加载 (Bulk Loading)

- 逐条 `INSERT` 效率极低。
- 应使用批量加载工具（如 MySQL `LOAD DATA INFILE`, Oracle `SQL*Loader`, Postgres `COPY`）。
- **主键生成**：
  - **Sequence** (Oracle, PG)：独立对象生成数字。
  - **Auto Increment / Identity** (MySQL, SQL Server)：列属性自增。

8 Chapter 8: 数据修改与函数 (Update, Delete, Functions)

8.1 8.1 UPDATE 与 DELETE

- **UPDATE**：
  - `UPDATE table SET col=val WHERE ...`
  - **致命错误**：忘记写 `WHERE` 子句会导致全表更新。
  - 使用子查询更新：`SET col = (SELECT ...)`。需确保子查询只返回一行，否则报错。
- **DELETE**：
  - `DELETE FROM table WHERE ...`
  - 逐行删除，记录日志，可回滚。
- **TRUNCATE**：
  - DDL 操作，快速清空全表，通常不可回滚（视 DBMS 实现），重置自增 ID。
- **MERGE (Upsert)**：
  - 标准语法 `MERGE INTO target USING source ON match ... WHEN MATCHED UPDATE ... WHEN NOT MATCHED INSERT ...`。
  - MySQL 使用 `INSERT ... ON DUPLICATE KEY UPDATE`。

8.2 8.2 存储函数与过程 (Functions & Procedures)

- **函数 (Function)**：必须有返回值，通常可用在 SQL 语句中（如 `SELECT my_func(col)`）。
- **过程 (Procedure)**：可以没有返回值，通过 `CALL` 或 `EXECUTE` 调用。
- **N+1 问题**：在 SELECT 列表中调用执行查询的函数，会导致每一行都执行一次查询，性能极差。应改写为 Join。

9 Chapter 9: 过程与触发器 (Procedures and Triggers)

9.1 9.1 存储过程优势

- **性能**：减少网络交互（一次调用，服务器端执行多步）。
- **安全**：可以只授予用户执行存储过程的权限，而不授予直接修改表的权限。
- **逻辑封装**：业务逻辑集中在数据库层。

9.2 9.2 触发器 (Triggers)

在特定的数据库事件（INSERT/UPDATE/DELETE）发生之前（BEFORE）或之后（AFTER）自动执行的代码。

- **用途**：
  - 强制复杂的完整性约束。
  - 审计（Auditing）：记录谁修改了数据。
  - 自动更新冗余数据（如更新文章表时自动更新分类表的文章计数）。
- **风险**：
  - **隐藏逻辑**：开发者可能不知道数据被触发器修改了。
  - **性能杀手**：大量行级触发器（FOR EACH ROW）会显著拖慢写入速度。
  - **死循环**：A 表触发器更新 B 表，B 表触发器更新 A 表。
- **建议**：尽量少用，除非无法通过约束或应用层逻辑解决。

10 Chapter 10: 存储与性能 (Storage and Performance)

10.1 10.1 物理存储结构

- **存储金字塔**：寄存器 > 缓存 > 内存 > SSD > HDD > 磁带。速度越快，价格越高，容量越小。
- **RAID**：磁盘阵列。

  - RAID 0：条带化，速度快，无冗余（坏一块盘全丢）。
  - RAID 1：镜像，安全，空间利用率 50%。
  - RAID 5/10：平衡方案。
- **块/页 (Block/Page)**：数据库 I/O 的最小单位（通常 4KB, 8KB, 16KB）。读取一行数据实际上通常会读取包含该行的整个页。

10.2 10.2 索引 (Indexing)

索引是加速数据检索的数据结构（类似于书的目录）。

- **B-Tree (Balanced Tree)**:最常用的索引结构。保持排序，适合范围查找 (`>`, `<`, `BETWEEN`) 和等值查找。
- **索引失效 (Pitfalls)**：

易错点 / Pitfall

以下情况可能导致索引失效（全表扫描）：1. **对索引列使用函数**：`WHERE YEAR(date_col) = 2020`。应改为 `WHERE date_col BETWEEN '2020-01-01' AND '2020-12-31'`。2. **隐式类型转换**：字符串列存数字，查询时用数字类型比较。3. **前导模糊查询**：`LIKE '%abc'`。4. **联合索引不满足最左前缀原则**：索引是 (A, B)，查询条件只有 B。

- **选择性 (Selectivity)**：索引在“区分度高”的列上最有效（如身份证号）。在“性别”这种只有两个值的列上建立索引通常无意义。
- **代价**：加快了读取，减慢了写入（INSERT/UPDATE/DELETE 需要维护索引树）。

10.3 10.3 执行计划 (Execution Plan)

- 使用 `EXPLAIN` 命令查看数据库如何执行 SQL。
- 关键看点：是否使用了索引 (Index Scan/Seek) 还是全表扫描 (Full Table Scan/Seq Scan)。