**2025-3-4 COMPSCI751 Summary**

**英文原文 & 中文翻译**

1. Introduction to the Course

(English)

• The database course covers foundational topics such as the Relational Model, Relational Algebra, SQL querying (from basic to advanced), storage and indexing, E-R (Entity-Relationship) modeling, normalization, query processing, query optimization, and transactions.

• The aim is to understand how databases manage massive amounts of data while providing persistence, multi-user support, efficiency, convenience, and reliability.

• A key aspect mentioned is that data typically outlives the database system itself, highlighting the importance of durability and proper design.

(Chinese)

1. 课程简介

• 本课程涵盖数据库的基础主题，包括关系模型、关系代数、SQL 查询（从基础到高级）、存储与索引、E-R（实体-关系）建模、数据库规范化、查询处理、查询优化以及事务处理等。

• 课程目标是理解数据库如何在处理海量数据的同时，提供持久性、多用户支持、高效性、方便性和可靠性。

• 课程中特别强调数据往往比数据库系统本身更“长寿”，因此持久化和恰当的数据库设计至关重要。

2. Relational Model and Schemas

(English)

• Relational Model: Data is organized into one or more “relations,” each represented as a table with rows (tuples) and columns (attributes).

• Schema: A relation’s schema defines its attributes and the domains (types) these attributes can have (e.g., string, integer, date).

• Keys:

• Primary Key: A minimal set of attributes uniquely identifying each tuple (e.g., ID in instructor).

• Foreign Key: An attribute in one relation that references a primary key in another relation (e.g., department\_name in instructor referencing department\_name in department).

• Null Values: Special marker indicating unknown or inapplicable data, which can complicate queries since null comparison logic differs from typical boolean comparisons.

Additional Notes / Expansion:

• Composite Keys: When more than one attribute is needed to form a unique identifier (e.g., (course\_id, sec\_id, semester, year) could be a composite key for a teaches relation).

• Superkeys & Candidate Keys: A superkey is any superset of attributes that uniquely identifies tuples. A candidate key is a minimal superkey (i.e., no proper subset of it is still a superkey).

(Chinese)

2. 关系模型与模式

• 关系模型：数据被组织为一个或多个“关系”，每个关系用一张表来表示，表由行（元组）和列（属性）构成。

• 模式（Schema）：一个关系的模式定义了它的属性以及这些属性可能具有的域（类型），例如字符串、整数或日期。

• 键（Keys）：

• 主键（Primary Key）：能够唯一标识每条元组的一组最小属性（比如 instructor 关系中的 ID）。

• 外键（Foreign Key）：一个关系中的某个属性，用来引用另一个关系的主键（例如 instructor 中的 department\_name 引用 department 中的 department\_name）。

• 空值（Null）：一种特殊标记，用于表示未知或不适用的数据；由于空值的比较逻辑与普通布尔比较不同，因此会给查询带来一定的复杂性。

补充与扩展：

• 复合键（Composite Key）：当需要多个属性组合在一起才能唯一标识元组时，比如在 teaches 关系中，(course\_id, sec\_id, semester, year) 就可能是一个复合键。

• 超级键（Superkey）与候选键（Candidate Key）：超级键指的是任意能够唯一标识元组的属性集合；候选键是其中最小的超级键（即去掉任何一个属性就不再是超级键）。

3. Relational Algebra

(English)

• Selection (σ): Filters rows based on a condition (e.g., \sigma\_{department\\_name = \text{Physics}}(\text{instructor})).

• Projection (π): Selects specific columns (attributes) from a relation (e.g., \pi\_{(ID,\ name,\ salary)}(\text{instructor})).

• Cross Product (×): Combines every tuple of one relation with every tuple of another, resulting in a large Cartesian product. Often used alongside a selection to “join” relevant tuples.

• Natural Join (⊲⊳): A shorthand for joining two relations on all common attribute names, requiring matching values in those attributes.

• Theta-Join (⋈\theta): A cross product plus a condition, often used when joining on specific expressions (e.g., instructor.ID = teaches.ID).

Additional Notes / Expansion:

• Composing Operators: We can nest these operators. For instance, to find the names (projection) of instructors (from a relation) in Physics (selection), we first do selection, then projection.

• Order of Operations: The order of selection and projection matters. Selecting first is typically more efficient and is necessary if we need columns (attributes) that might otherwise be projected away.

Relational Algebra Pseudocode Example:

# Find the names of instructors in the Physics department

π name ( σ department\_name = "Physics" ( instructor ) )

# Equivalently, to join instructors with teaches only if they match on ID

σ instructor.ID = teaches.ID ( instructor × teaches )

# or using natural join:

instructor ⊲⊳ teaches # if they share an attribute named ID

(Chinese)

3. 关系代数

• 选择（σ）：根据条件筛选行（例如 \sigma\_{department\\_name = \text{Physics}}(\text{instructor})）。

• 投影（π）：选取关系中的特定列（属性）（例如 \pi\_{(ID,\ name,\ salary)}(\text{instructor})）。

• 笛卡儿积（×）：将一个关系的每条元组与另一个关系的每条元组配对，得到大规模的结果集，通常需要配合选择来筛选匹配元组。

• 自然连接（⊲⊳）：用于在两个关系的公共属性上进行连接，要求公共属性的取值相同。

• θ-连接（⋈\theta）：笛卡儿积与条件相结合的形式，当需要指定特定表达式（如 instructor.ID = teaches.ID）时常用。

补充与扩展：

• 运算符组合：可以将不同的关系代数运算符组合在一起使用。例如，想要找到物理系教师的姓名时，先做选择（找出物理系），再投影（只保留姓名）。

• 操作顺序：选择与投影的顺序至关重要。先做选择往往更高效，如果先投影可能会丢掉之后选择需要用到的列。

关系代数伪代码示例:

# 查找物理系所有教师的姓名

π name ( σ department\_name = "Physics" ( instructor ) )

# 若要连接 instructor 与 teaches 并匹配它们的 ID

σ instructor.ID = teaches.ID ( instructor × teaches )

# 或使用自然连接：

instructor ⊲⊳ teaches # 如果它们都有名为 ID 的公共属性

4. SQL Querying (Brief Overview)

(English)

• Basic SQL Structure:

SELECT column\_list

FROM table\_list

WHERE condition;

• The FROM clause conceptually performs a Cartesian product among the listed tables.

• The WHERE clause then filters the rows.

• The SELECT clause picks which columns to output.

• Further Topics: grouping (GROUP BY), aggregation (COUNT, SUM, AVG, etc.), subqueries, joins, set operations (UNION, INTERSECT, EXCEPT), etc.

Code Example (SQL):

-- Find the names of instructors in Physics with salary above 90000

SELECT name

FROM instructor

WHERE department\_name = 'Physics'

AND salary > 90000;

(Chinese)

4. SQL 查询（简要概述）

• 基本 SQL 结构：

SELECT column\_list

FROM table\_list

WHERE condition;

• 在概念上，FROM 子句会对所列出的表做笛卡儿积。

• WHERE 子句用于筛选行。

• SELECT 子句指定最终要输出哪些列。

• 进阶主题：分组（GROUP BY）、聚合函数（COUNT、SUM、AVG 等）、子查询、连接、集合操作（UNION、INTERSECT、EXCEPT）等。

SQL 示例代码:

-- 查找物理系且薪资高于 90000 的教师姓名

SELECT name

FROM instructor

WHERE department\_name = 'Physics'

AND salary > 90000;

5. E-R Model and Normalization

(English)

• E-R Model:

• Entities represent objects (e.g., Student, Instructor), with attributes describing them.

• Relationships link entities (e.g., “Student takes Course”).

• Used to create a high-level design before transforming into relational schemas.

• Normalization:

• A process of organizing attributes and tables to reduce data redundancy and improve data integrity.

• Normal Forms (1NF, 2NF, 3NF, BCNF, etc.) define levels of “good” design.

• Additional Notes / Expansion:

• Functional Dependencies are key to understanding how attributes relate and where redundancies might form.

• ER-to-Relational Mapping typically involves creating a table for each strong entity and possibly separate tables for relationships or weak entities, depending on cardinality constraints.

(Chinese)

5. E-R 模型与数据库规范化

• E-R 模型：

• 实体用于表示对象（例如 Student、Instructor），属性描述实体本身。

• 关系用于连接实体（例如“学生选修课程”）。

• 在将设计转换为关系模式之前，E-R 模型用来绘制较高层次的系统结构。

• 规范化：

• 通过将属性和表结构进行合理组织，以减少数据冗余并提高数据完整性。

• 不同的范式（1NF、2NF、3NF、BCNF 等）衡量设计的合理程度。

• 补充与扩展：

• 函数依赖（Functional Dependencies） 是理解属性间相互关系、查找冗余的关键。

• E-R 到关系模式的映射：通常给每个强实体创建一张表，对于关系或弱实体（取决于基数约束）也可能单独建表。

6. Data Storage & Indexing (Brief Expansion)

(English)

• Physical Storage: Databases store data on disks or SSDs, often organizing it into pages. Minimizing disk I/O is crucial for performance.

• Indexing:

• B+ Trees: Common for range queries. Balanced tree structure for efficient insert, delete, and search in O(log n) time.

• Hash Indexes: Useful for exact matches (equality searches), less suited for range queries.

• Additional Notes:

• The teacher mentioned that storage and indexing create the “illusion” of infinite and persistent storage for users. This involves complexities like buffer management, caching, concurrency on the storage layer, etc.

(Chinese)

6. 数据存储与索引（简要扩展）

• 物理存储：数据库将数据存放在磁盘或 SSD 上，一般以页面（page）为单位进行组织。减少磁盘 I/O 对性能至关重要。

• 索引：

• B+ 树：最常见的索引结构，适用于区间查询。由于是平衡树结构，插入、删除和查找通常在 O(log n) 时间内完成。

• 哈希索引：更适用于等值匹配查询，不太适合区间（范围）查询。

• 补充说明：

• 老师提到，存储和索引为用户打造了一个看似“无限且持久”的存储环境，这背后涉及诸如缓冲管理、缓存以及存储层的并发等复杂问题。

7. Transactions & Concurrency (Expansion)

(English)

• Transaction: A sequence of operations performed as a single logical unit of work—must either complete in its entirety or not at all (atomicity).

• ACID Properties:

1. Atomicity: The whole transaction succeeds or fails completely.

2. Consistency: The transaction takes the database from one valid state to another valid state.

3. Isolation: Concurrent transactions appear as if they were executed in sequence (no interference).

4. Durability: Once a transaction commits, its effect remains even in case of failures.

• Concurrency Control (briefly mentioned in class): Mechanisms like Two-Phase Locking (2PL) or Multiversion Concurrency Control (MVCC) to ensure transactions do not conflict.

Additional Notes / Expansion:

• Isolation Levels: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE. These define how “strictly” the system prevents phenomena like dirty reads, non-repeatable reads, and phantom reads.

• Example: If transferring $100 from Account A to Account B, either both debit and credit happen (commit) or none happens (rollback).

(Chinese)

7. 事务与并发（扩展）

• 事务（Transaction）：一系列操作作为一个逻辑整体执行——要么全部成功，要么全部失败（原子性）。

• ACID 特性：

1. 原子性（Atomicity）：事务要么全部执行完毕，要么完全不执行。

2. 一致性（Consistency）：事务会将数据库从一个合法状态转变为另一个合法状态。

3. 隔离性（Isolation）：并发事务在逻辑上看起来像是顺序执行，互不干扰。

4. 持久性（Durability）：一旦事务提交，其效果不会因故障而丢失。

• 并发控制（课堂上简要提到）：诸如 两阶段锁协议（2PL） 或 多版本并发控制（MVCC） 等机制，用来确保并发事务之间不会互相冲突。

补充与扩展：

• 隔离级别：READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ、SERIALIZABLE。这些定义了系统在多大程度上防止脏读（dirty read）、不可重复读（non-repeatable read）以及幻读（phantom read）。

• 示例：从账户 A 转 $100 到账户 B，要么借记和贷记都执行（提交），要么都不执行（回滚）。

Summary and Review

(English)

• Databases are used everywhere: from credit card transactions to social media, from email systems to enterprise accounting. They handle massive data volumes, offer concurrent access, ensure persistence, and provide convenience to end users.

• The Relational Model and its formal foundation in Relational Algebra pave the way for SQL. Understanding the basic operations (selection, projection, join) is crucial.

• Database design involves conceptual modeling (E-R model), mapping to relational schemas, and optimizing using normalization. Physical aspects of data storage and indexing ensure efficient access.

• Finally, transactions guarantee atomic and consistent modifications to data under concurrent usage, highlighting the importance of concurrency control.

(Chinese)

总结与复习

• 数据库无处不在：从信用卡交易到社交媒体，从邮件系统到企业财务，处理海量数据并支持多用户并发访问，同时保证数据的持久性与用户操作的便利性。

• 关系模型及其在关系代数中的形式化基础为 SQL 奠定了理论根基。掌握选择、投影和连接等基本操作至关重要。

• 数据库设计通常从 E-R 概念建模开始，然后映射到关系模式，并通过规范化进行优化。数据的物理存储与索引则保证了数据访问的效率。

• 最后，事务为并发环境下的数据修改提供原子性与一致性保证，并发控制尤为重要。