# Indexes, Views, Triggers, Stored Procedures and Functions

**1. Indexes 索引**

* **定义**：索引是附属于表或视图的数据结构，其主要目的是**加速数据检索**。
* **特点**：
  + 可以建立在单列或多列上。
  + 主键（Primary Key）默认会自动带有索引。
* **适用场景**：
  + WHERE 条件过滤
  + JOIN 操作时加速匹配
  + 聚合查询（MAX、MIN、GROUP BY、HAVING）
  + 排序、分组
* **优点**：提高查询速度。
* **缺点**：插入、删除、更新操作时会有额外维护开销。

**2. Views 视图**

* **定义**：视图是基于 SQL 查询的 **虚拟表**，其内容（行和列）由查询结果定义。
* **用途**：
  + 提供特定用户的定制化数据视图（例如员工只能看到自己部门的数据）。
  + 出于安全考虑，不让用户直接访问底层表。
  + 当表结构变化但需要保持前端兼容时，使用视图作为“过渡层”。
* **特点**：
  + 默认视图不存储数据，查询时实时计算。
  + 如果存储（Materialized View 物化视图），需要维护数据同步。
* **优缺点**：
  + 优点：简化查询、提高安全性。
  + 缺点：普通视图性能依赖底层表，可能较慢。

**3. Temporary Tables 临时表**

* **定义**：在会话中临时存在的表，会话结束后自动删除。
* **用途**：
  + 提供一个临时的工作空间存储和操作数据。
  + 和视图不同，临时表的数据是存储的，而不是实时计算的。
* **创建方式**：
  + CREATE TEMPORARY TABLE ... （语法与普通表类似，多了 TEMPORARY 关键字）。
  + CREATE TEMPORARY TABLE AS SELECT ... （直接用查询结果生成临时表）。
* **优点**：灵活处理临时数据，不影响数据库原始表。

**4. Triggers 触发器**

* **定义**：触发器是数据库中 **自动执行的程序逻辑**，当某些事件发生时触发。
* **触发事件**：
  + BEFORE INSERT / AFTER INSERT
  + BEFORE UPDATE / AFTER UPDATE
  + BEFORE DELETE / AFTER DELETE
* **用途**：
  + 自动维护数据完整性（例如：插入评分时自动更新教师的总分）。
  + 实现复杂业务逻辑。
  + 监控表数据变化。
* **优点**：自动化、减少应用端逻辑。
* **缺点**：调试难度大，逻辑复杂时可能影响性能。

**5. Stored Procedures 存储过程**

* **定义**：存储过程是数据库中 **已命名的 SQL 语句序列**，相当于数据库中的“方法/函数”。
* **用途**：
  + 封装复杂或频繁使用的查询逻辑。
  + 保证业务逻辑一致性（逻辑集中存放）。
* **优点**：
  + 减少应用与数据库之间的通信量。
  + 集中化逻辑，避免多处代码冗余。
* **缺点**：
  + 占用数据库服务器资源，调试和维护相对困难。

**6. Functions 函数**

* **定义**：函数（通常是用户自定义函数 UDF）类似存储过程，但通常 **返回一个值或一个结果集**。
* **用途**：
  + 可以像内置函数一样在查询中使用。
  + 例如：传入某个课程 ID，返回选修该课程的学生数量。
* **特点**：
  + 可以有输入参数。
  + 必须声明 **是否确定性（deterministic）**，确保相同输入得到相同结果。
* **与存储过程的区别**：
  + 存储过程通常是执行一系列操作，结果可能是多个查询或更新。
  + 函数更倾向于返回一个单一值，可直接嵌入 SQL 查询中。

# Advanced Window Functions and OLAP Queries

**🔹 第一部分：高级窗口函数（Advanced Window Functions）**

**排名函数**

1. **RANK()**
   * 按顺序给行编号。
   * 如果有并列值，排名相同，但中间会跳号。
   * 例子：工资并列 → 排名结果可能是 1, 1, 3。
2. **DENSE\_RANK()**
   * 与 RANK 类似，但不会跳号。
   * 例子：工资并列 → 排名结果可能是 1, 1, 2。
3. **ROW\_NUMBER()**
   * 每一行一个唯一编号，不管值是否重复。
   * 例子：工资并列 → 排名结果可能是 1, 2, 3。
4. **NTILE(n)**
   * 把结果集尽量均匀分成 n 组（桶）。
   * 例子：NTILE(4) → 把 20 行数据分成 4 组，每组大约 5 行。
5. **PERCENT\_RANK()**
   * 百分比排名。
   * 计算方法：(当前行的排名 - 1) / (总行数 - 1)。
   * 结果范围：0 到 1。第一个值是 0，最后一个值是 1。
6. **CUME\_DIST()**（累计分布）
   * 计算小于等于当前行值的行数占总行数的比例。
   * 结果范围：0 到 1。最后一行一定是 1。

**窗口与帧（Window / Framing）**

窗口函数可以在不折叠数据的情况下，做滑动计算。

常用边界：

* UNBOUNDED PRECEDING → 从第一行到当前行。
* CURRENT ROW → 当前行。
* n PRECEDING → 往前 n 行。
* n FOLLOWING → 往后 n 行。

**常见应用：**

* 移动平均（如三天平均）。
* 运行总和（累积和）。

**🔹 第二部分：OLAP 查询（OLAP Queries）**

**OLAP 基础**

* OLAP（联机分析处理）主要用于商业智能，分析历史数据和多维度数据。
* 与 OLTP（联机事务处理）不同：
  + OLTP → 日常交易（超市收银）。
  + OLAP → 报表、趋势分析。

**两个关键概念：**

* 维度属性：如城市、季度、产品类别。
* 度量属性：如销售额，需要聚合。

**GROUP BY, ROLLUP, CUBE**

1. **GROUP BY**
   * 按字段分组并聚合。
   * 只返回分组结果，没有小计或总计。
2. **ROLLUP**
   * 在 GROUP BY 的基础上，自动增加小计和总计。
   * 如果 GROUP BY 有 n 个字段，ROLLUP 会产生 n+1 个层次。

例子：

* + GROUP BY city, quarter
  + ROLLUP(city, quarter) → 会额外生成：
    - 每个城市的小计
    - 所有城市的总计

1. **CUBE**
   * 生成所有可能的分组组合。
   * 如果有 n 个字段，CUBE 会产生 2^n 种组合。
   * 适合多维分析，比如同时看“城市 + 产品 + 季度”的所有可能组合。

# Database Application Development

**1. JDBC（Java Database Connectivity）**

* **作用**：Java 提供的 API，用于与数据库系统交互。
* **功能**：不仅能执行 SQL，还能获取数据库的元数据（表、字段、类型）。
* **通信模型**：
  1. 建立连接（Connection）
  2. 创建语句对象（Statement）
  3. 执行 SQL（Execute）
  4. 获取结果集（ResultSet）
  5. 异常处理（try-catch）
  6. 关闭资源（Connection、Statement、ResultSet）

**2. JDBC 基本代码结构**

* 使用 DriverManager.getConnection() 建立连接。
* 使用 createStatement() 创建语句对象。
* 使用 executeQuery() 或 executeUpdate() 执行 SQL。
* 使用 ResultSet 遍历数据（next() 移动游标，getInt / getString / getDouble 获取字段值）。
* 最后关闭连接和对象。

**3. PreparedStatement（预处理语句）**

* **优势**：
  + 可复用，性能更好。
  + 支持参数化查询（避免频繁写静态 SQL）。
  + 防止 SQL 注入攻击（自动处理转义字符）。
* **用法**：
  + ? 作为参数占位符。
  + 使用 setInt()、setString() 等方法绑定参数。

例子：

String sql = "SELECT \* FROM Students WHERE score >= ? AND topic = ?";

PreparedStatement pstmt = conn.prepareStatement(sql);

pstmt.setInt(1, 109);

pstmt.setString(2, "Java");

ResultSet rs = pstmt.executeQuery();

**4. CallableStatement（调用存储过程/函数）**

* 用于调用数据库中的 **存储过程** 和 **函数**。
* 使用 prepareCall() 方法。
* 可以传入多个参数（比如学生 ID、课程 ID、分数等）。
* 如果过程修改表，用 executeUpdate()；如果返回结果，用 executeQuery()。

**5. SQL 注入与防护**

* **风险**：用户恶意输入 SQL 片段操纵数据库。
* **防护**：使用 PreparedStatement 自动转义参数，避免拼接字符串。

**6. 其他语言的数据库连接**

除了 Java，Python 和 C++ 也有类似模型：

* **Python**：mysql.connector → connect() → cursor() → execute() → fetchall()
* **C++**：通过指针对象连接数据库，执行 SQL，遍历 ResultSet，最后释放资源。

# Indexing and Hashing

**📌 索引 (Indexing)**

**1. 索引的作用**

* 索引是一种 **数据结构**，用来加快数据库中数据的检索速度。
* 索引文件只保存 **搜索键 (search key) + 指针 (pointer)**，因此比原始文件小。

例子：

* 学生表中的 SID 可以作为索引。
* 评分表中 SID + PID 的组合键也可以作为索引。

**2. 索引的分类**

* **主索引 (Primary Index)**
  + 搜索键和数据文件中的记录顺序一致（有序存储，聚簇索引）。
  + 一般是主键，但不一定必须是主键。
* **辅助索引 (Secondary Index)**
  + 搜索键的顺序与数据文件存储顺序不同（非聚簇索引）。

**3. 索引的存储方式**

* **稠密索引 (Dense Index)**
  + 每个搜索键在索引中都有一条记录。
  + 插入/删除时需要更新所有对应索引。
* **稀疏索引 (Sparse Index)**
  + 每个数据块（block）的第一个键值出现在索引中。
  + 插入/删除只在新块创建时更新索引，更节省空间。

**4. 多级索引 (Multilevel Indexing)**

* 如果单级索引太大放不进内存，可以建立多级索引。
* 例如：**一级稀疏索引** + **二级稠密索引**。
* 常见实现就是 **B+ 树**。

**📌 B+ 树 (B+ Tree)**

**1. 特点**

* 一种 **自平衡树**，保证所有叶子节点在同一层。
* 每个节点存储最多 n 个键值和 n+1 个指针。
* 叶子节点存储实际记录指针，内部节点只存搜索键。
* **50% 填充率原则**：保证节点至少半满，避免极端不平衡。

**2. 基本操作**

* **搜索**：从根节点开始，逐层根据键值范围选择指针，直到叶子节点。
* **插入**：
  + 如果叶子有空位 → 直接插入并保持有序。
  + 如果满了 → **分裂 (split)** 节点，并将中间键值上移。
  + 如果父节点也满 → 父节点继续分裂，可能导致新根节点产生。
* **删除**：
  + 删除后节点仍大于等于半满 → 直接删除。
  + 如果不足半满 → 向兄弟节点借（redistribute）。
  + 如果兄弟也不足 → 合并 (coalesce/merge)，并修改父节点。

**📌 哈希 (Hashing)**

**1. 静态哈希 (Static Hashing)**

* 使用哈希函数将搜索键映射到 **固定数量的桶 (buckets)**。
* 常用公式： h(K) = K mod B，其中 B = 桶数。
* 字符串键：转成二进制求和再取模。

**2. 问题**

* 桶数固定 → 当数据量增加时容易产生 **溢出 (overflow)**。
* 解决方法：**溢出链 (overflow chaining)**，形成链表。

**3. 动态哈希 (Dynamic Hashing)**

* 根据数据规模动态调整桶的数量（例如 Extendible Hashing、Linear Hashing）。
* 可以避免静态哈希的溢出问题。

# Transaction

**📌 什么是事务 (Transaction)**

* **事务**：一组数据库操作（读/写）组成的逻辑单元，要么全部执行，要么全部不执行。
* 操作类型：
  + **Read(A)**：读取数据元素 A
  + **Write(A)**：写入数据元素 A
  + **Commit**：提交（更改永久生效）
  + **Abort**：中止（撤销事务，回滚）

**📌 ACID 特性**

1. **Atomicity（原子性）**
   * 全做或全不做，没有“部分执行”。
   * 失败则回滚。
2. **Consistency（一致性）**
   * 执行前后，数据库必须保持一致性约束。
   * 例如转账前后，A+B 的总和不变。
3. **Isolation（隔离性）**
   * 并发执行的事务之间互不干扰。
   * 每个事务好像是独立顺序执行的。
4. **Durability（持久性）**
   * 一旦提交，结果永久保存在数据库，即使系统崩溃。

**📌 事务状态**

* **Active**：正在执行
* **Partially Committed**：执行了最后一条语句，但还没完全提交
* **Committed**：提交完成，结果持久
* **Failed**：事务出错
* **Aborted**：已回滚

**📌 并发与调度 (Schedules)**

* **Serial Schedule**：事务一个一个顺序执行
* **Interleaved Schedule**：事务交叉执行（常见于并发环境）
* **Serializable Schedule**：调度的执行结果与某个串行调度等价

**📌 可串行化 (Serializability)**

* **Conflict Serializability（冲突可串行化）**
  + 如果通过交换**不冲突操作**，能得到某个串行调度 → 可串行化。
  + 冲突条件：两个操作满足
    1. 操作属于不同事务
    2. 访问同一数据项
    3. 至少有一个是写
  + 例子：
    1. **Read–Read**：不冲突
    2. **Read–Write** / **Write–Read** / **Write–Write**：冲突
* **判定方法**：
  + 构造 **优先图 (Precedence Graph)**：
    1. 节点 = 事务
    2. 边 = 冲突的先后依赖
  + **无环 (Acyclic)** → 冲突可串行化
  + **有环 (Cycle)** → 不可串行化

**📌 恢复性 (Recoverability)**

* **Recoverable Schedule**：
  + 如果事务 Ti 读取了 Tj 的写结果 → Tj 必须在 Ti 提交之前提交。
* **Cascading Rollback（级联回滚）**：
  + 一个事务失败导致依赖它的其他事务也要回滚。
* **Cascadeless Schedule**：
  + 不允许事务读取尚未提交的数据 → 避免级联回滚。
  + 所有 cascadeless 调度都是 recoverable。

# Concurrency Control

**📌 并发控制的动机**

* **目标**：在多个事务同时运行时，保证数据库的一致性。
* 我们希望调度（Schedule）是：
  + **Conflict Serializable（冲突可串行化）**
  + **Recoverable（可恢复的）**
  + **最好是 Cascadeless（无级联回滚）**

**📌 两种并发控制思想**

1. **Pessimistic Concurrency Control（悲观并发控制）**
   * 在访问数据前先加锁，防止冲突。
   * 适用于冲突常见的场景。
2. **Optimistic Concurrency Control（乐观并发控制）**
   * 不事先加锁，先并发执行，直到提交时再检查冲突。
   * 适用于冲突少的环境。

**📌 锁机制 (Locks)**

* **锁的作用**：限制事务对数据项的访问，保证一致性。
* **两种锁：**
  + **S 锁 (Shared Lock，共享锁)**
    - 只允许读，不允许写。
    - 多个事务可以同时持有 S 锁。
  + **X 锁 (Exclusive Lock，排他锁)**
    - 允许读和写。
    - 一次只能有一个事务持有，禁止其他事务访问。
* **兼容性**：
  + S + S → ✅ 兼容
  + S + X → ❌ 冲突
  + X + X → ❌ 冲突

**📌 两段锁协议 (Two-Phase Locking, 2PL)**

* **核心规则**：事务分为两个阶段：
  1. **扩展阶段（Growing phase）**：只申请锁，不释放锁
  2. **收缩阶段（Shrinking phase）**：只释放锁，不再申请新锁
* 保证 **可串行化**，但仍可能导致死锁。
* **变体：**
  1. **Strict 2PL**：X 锁直到事务提交后才释放 → 避免级联回滚
  2. **Rigorous 2PL**：所有锁（S 和 X）直到提交后才释放 → 更严格

**📌 死锁 (Deadlock)**

* **定义**：一组事务相互等待对方持有的锁，无法继续执行。
* **解决方法**：
  1. **检测**：利用等待图（Wait-For Graph）发现循环依赖。
  2. **恢复**：选择“牺牲者”事务（Victim）回滚，释放锁。
     + 通常选择代价最小的事务（例如执行最少操作的）。
* **可能问题**：反复回滚同一事务 → **饥饿（Starvation）**。

**📌 锁管理器 (Lock Manager)**

* 作为一个独立模块：
  + 负责记录当前哪些锁已被授予，哪些在等待队列。
  + 使用 **锁表（Lock Table）** 管理数据项和锁请求。
  + 处理事务请求：授予锁 / 让事务等待 / 让事务回滚。

**📌 小结**

* **并发控制的目标**：在保证高吞吐量的同时，保持数据库一致性。
* **主要技术**：基于锁的协议（2PL 及其变体）。
* **挑战**：死锁、饥饿、回滚。
* **解决之道**：使用锁管理器 + 死锁检测/恢复策略。

⚡ 一个直观的类比：

* **共享锁 (S 锁)** 就像图书馆的一本参考书，可以多人同时看，但不能带走。
* **排他锁 (X 锁)** 就像借书卡，只能一个人借走，别人必须等他还书。
* **死锁** 就像两个人互相拿着对方需要的书，都不愿意先放下 → 谁都学不了。
* **解决办法**：老师强制一个人先把书放回去（回滚），这样另一个人能继续。

# Summary

## Module 1 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Explain the purpose of indexes in a database.
* Describe how indexes improve query performance.
* Create and drop indexes in a relational database.
* Create simple and complex views that present data from one or more tables.
* Explain the benefits of using views for data abstraction.
* Update, delete, and insert data through views.
* Describe the limitations and considerations when modifying views.
* Create and use user-defined functions in SQL.
* Describe common built-in functions in SQL for string manipulation, date and time operations, and mathematical calculations.
* Create triggers that automatically execute SQL statements in response to specified events.
* Discuss best practices and potential pitfalls when using triggers.
* Identify scenarios where triggers are useful and where they should be avoided.
* Create temporary tables to store intermediate or session-specific data.
* Insert, update, and delete data from temporary tables.
* Create stored procedures that encapsulate sets of SQL statements.
* Use parameters to pass data to and from stored procedures.
* Describe the benefits of using stored procedures for code modularity and reusability.

## Module 2 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Define what window functions are in SQL and their benefit in data analysis.
* Describe advanced applications of a variety of window functions for ranking and partitioning data, such as advanced usage of RANK(), DENSE\_RANK(), LEAD(), LAG(), NTILE, PERCENT\_RANK, and CUME\_DIST().
* Use PARTITION BY and ORDER BY clauses to control window function behavior.
* Describe window frames and their role in window functions.
* Use RANGE and ROWS clauses for defining custom window frames.
* Describe the fundamental concepts of OLAP, including dimensions, measures, and hierarchies.
* Define dimensions, measures, and aggregations using GROUP BY, ROLLUP, CUBE, and GROUPING SETs
* Identify the differences between OLAP and OLTP (Online Transaction Processing) databases.
* Engage in hands-on projects and case studies that involve designing, querying, and analyzing OLAP cubes.
* Apply advanced window functions and OLAP queries to real data sets.

## Module 3 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Develop proficiency developing database applications using Java, Python, or C++
* Establish connections to databases from using programming languages that support SQL.
* Perform CRUD (Create, Read, Update, Delete) operations on the database using programming languages.
* Write code using Java, Python or C++ to insert, retrieve, update, and delete data records.
* Write code to support complex SQL features, as such triggers, functions, and stored procedures.
* Implement error and exception handling to ensure robust database interactions.
* Handle database connection errors, query failures, and data validation issues.
* Work on real-world database application projects to apply knowledge and skills.
* Solve database-related challenges commonly encountered in industry.

## Module 4 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Define indexing and its importance in database systems.
* Differentiate between types of indexes, including ordered, hash index.
* Differentiate between primary and secondary, multi-level and single-level index.
* Describe the structure of a B+-tree and its components, including nodes and leaves.
* Explain how B+-trees maintain balance and support efficient search operations.
* Explain how a B+- tree insertion and deletion operations work.
* Explain the concept of hashing (static) and its role in data retrieval.
* Describe how hash functions work.

## Module 5 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Explain transactions in databases and their importance in ensuring data consistency and integrity.
* Describe the four ACID properties (Atomicity, Consistency, Isolation, Durability) and their significance in transactions.
* Identify techniques for managing concurrent access to data.
* Discuss the COMMIT and ROLLBACK behavior in concurrent transactions.
* Define the serial schedules, serializable schedules, and concept of serializability in the context of concurrent transactions.
* Explain how to test if a given set of transactions is conflict-serializable using graph-based techniques.
* Apply the precedence graph test and cycle detection to determine conflict serializability.
* Describe the concept of equivalent serial schedules and their relation to conflict serializability.
* Identify schedules that produce the same final database state.
* Identify whether a schedule is recoverable and cascade less.

## Module 6 Summary

After completing all of the materials, please take a moment to reflect on what was covered. The learning objectives below capture the key takeaways from this module:

* Define concurrency control in the context of database management systems.
* Describe the impact of uncontrolled concurrent access on data integrity.
* Describe the principles of lock-based concurrency control.
* Describe lock types (shared and exclusive), lock modes, and lock compatibility.
* Explain the two-phase locking protocol and its guarantees.
* Explain how transactions acquire and release locks in two-phase locking.
* Define deadlocks and discuss their causes.