Normalization & Transaction

1. Normalization

1.1. Normalization定义

1️⃣定义

	操作	目的
Normalization	一大表格$\xrightarrow{\text{拆分}}$若干小表格	去除Redundancies从而避免Anomoly
Denormalization	一大表格$\xleftarrow{\text{拼接}}$若干小表格	避免频繁的Join

⚠️Fully Normalized = Eliminate All Anomalies，一般到4NF才算完全归一化

2️⃣示例

1. 标准化前

StuID	SName	CID	CName
1	John	101	Math
2	Mary	102	English
1	John	103	Physics
3	Peter	101	Math
2	Mary	103	Physics

2. 标准化后

学生表	StuID	SName	课程表	CID	CName	学生-课程表	StuID	CID
\	1	John	\	101	Math	\	1	101
\	2	Mary	\	102	English	\	2	102
\	1	John	\	103	Physics	\	1	103
\	3	Peter	\	101	Math	\	3	101
\	2	Mary	\	103	Physics	\	2	103

3️⃣Normalization/Denormalization对比

操作	Normalization，比如 A, B	Denormalization，比如 $A \bowtie B$
查询一个表	快	慢
查询两个表	慢	快
插入/修改/删除	容易	困难

1.2. Database Anomalies异常

1️⃣ 概述：期末会给你表格，让你用文字描述举例三种Anomaly

1. 示例表格

Order(PFK)	Item(PFK)	Desc	Qty
27	873	nut	2
28	873	nut	10
30	495	washer	50

2. 基本概念

Anomaly	含义	示例(下表)
Insertion	插入数据→插入其他数据	插入Item/Desc $\xrightarrow{\text{还需}}$插入Order/Qty
Deletion	删除数据→删除其他数据	删除Order30 $\xrightarrow{\text{引起}}$整行删除
Update	更新数据→更新其他数据	更新Item873的Desc $\xrightarrow[\text{有两处Item873}]{\text{引起}}$两处Desc更新

2️⃣ 用NULL来解决Anomaly: 不能根本解决

1. Insertion Anomaly可能可以，原因在于PK是不能NULL的
   - 可以的示例：插入Order/Item/Desc → Qty为NULL
   - 不可以的示例：插入Item/Desc → Qty为NULL/Order不能为NULL
2. Deletion Anomaly: 同理可能可以，当删除的数据不是PK时，可考虑设为NULL
3. Update Anomaly: 绝对没戏，不可能更新成NULL

1.3. Functional Dependencies

1️⃣ 概述

1. 定义：A set of attributes X determines another set of attributes Y uniquely
2. 符号：X → Y表示如果知道X就知道Y，比如Stu-Name → Stu-FirstName
3. 性质：
   - 依赖恒等式：$X \to X$，PK → All Column
   - 拆分原则：$AB \to CD$ $\begin{cases} \text{可以拆分为: } AB \to C \ / \ AB \to D \\ \text{不可拆分为: } A \to CD \ / \ B \to CD \end{cases}$ → 只有右侧可拆分

2️⃣ 有关概念：以A($\underline{X}, \underline{Y}, Z, D$)为例

1. Determinants:
   - 箭头左侧的属性，比如$\underline{X}, \underline{Y} \to Z$中的XY
   - 至于这其中Z叫什么就无具体说法
2. Key/Non-Key属性：包含/不包含PK的属性，比如

属性类型	含义	示例
Key属性	包含PK的属性，或者说是PK的一部分	$\underline{X}, \underline{Y}$
Non-Key属性	其余属性	Z, D

3. 两种Dependency

依赖类型	含义	示例
Partial Functional Dependency	Key属性(PK的一部分) $\xrightarrow{\text{决定}}$ Non-Key属性	$\underline{Y} \to Z$
Transitive Dependency(传递性)	Non-Key属性 $\xrightarrow{\text{决定}}$ Non-Key属性	$Z \to D$

3️⃣ Armstrong's Axioms(公理)

Axioms	条件	⇒	结果	备注
Reflexivity	$B \subseteq A$	$\Rightarrow$	$A \to B$	或者说$AB \to A$
Augmentation(扩展)	$A \to B$	$\Rightarrow$	$AC \to BC$	N/A
Transitive	$A \to B \ \wedge \ B \to C$	$\Rightarrow$	$A \to C$	N/A

1.4. Normal Form

0️⃣ Overview

Normal Form	Description
0NF	混沌之物 (Non-Relational)
1NF	无Repeating Groups
2NF	无Repeating Groups+Partial Dependency
3NF	无Repeating Groups+Partial Dependency+Transitive Dependency

1️⃣ 1NF

1. 1NF 的基本特征，(不满足以下特征的就是Non-Relational Model)
   - 不能有重复的列
   - 不能有Repeating Groups，即表中一个单元(Fill)只能有一个数据
     (不同单元的数据用逗号隔开，但注意有些数据单元内本就有逗号如Address)
2. Non-Relational Model→1NF:
   - 将Repeating Groups从原表中拆分成新表
   - 将原表的PK $\begin{cases} \text{一份保留在原关系} \to \text{作PK} \\ \text{一份复制到新关系} \to \text{作PFK} \end{cases}$
3. 示例：注意区分$\underline{\text{PK}}$，$\overline{\text{FK}}$，$\overline{\underline{\text{PFK}}}$的表示方法

$$\text{Order-Item}(\underline{\text{Order}}, \text{Cust}, (\underline{\text{Item}}, \text{Desc}, \text{Qty})) \to \begin{cases} \text{Order}(\underline{\text{Order}}, \text{Cust}) \\ \text{Order-Item}(\overline{\underline{\text{Order}}}, \underline{\text{Item}}, \text{Desc}, \text{Qty}) \end{cases}$$

2️⃣ 2NF

1. 2NF基本特征：Part of Composite Key $\xrightarrow[\times]{\text{Identify}}$ Non-key Attribut (NO Partial 依赖)
2. 1NF→2NF: 移除Partial Dependencies
   - 把Partial Dependencies从原关系中剥离出来
   - Partial Dependencies的Determinant $\begin{cases}\text{一份保留在原关系} \rightarrow \text{作PFK}\\ \text{一份剥离到新关系} \rightarrow \text{作PK}\end{cases}$
3. 示例：注意区分$\underline{\text{PK}}$，$\overline{\text{FK}}$，$\overline{\underline{\text{PFK}}}$的表示方法
   - 假设Order-Item($\overline{\underline{\text{Order}}}$, $\underline{\text{Item}}$, Desc, Qty))存在 $\begin{cases}\overline{\underline{\text{Order}}}, \underline{\text{Item}} \rightarrow \text{Qty}\\ \underline{\text{Item}} \rightarrow \text{Desc (Partial Dependency)}\end{cases}$
   - Order-Item→ $\begin{cases}\text{Order}(\underline{\text{Order}}, \text{Cust})\\ \text{Order-Item}(\overline{\underline{\text{Order}}}, \underline{\text{Item}}, \text{Desc}, \text{Qty}))\rightarrow \begin{cases}\text{Item}(\underline{\text{Item}}, \text{Desc})\\ \text{Order-Item}(\overline{\underline{\text{Order}}}, \overline{\underline{\text{Item}}}, \text{Qty})\end{cases}\end{cases}$

3 3NF

1. 3NF基本特征：Non-key Attribut $\xrightarrow[\times]{\text{Identify}}$ Non-key Attribut (NO Transitive 依赖)
2. 2NF→3NF: 移除Transitive Dependencies
   - 把Transitive Dependency从原关系剥离出来
   - Transitive Dependency的Determinant $\begin{cases}\text{一份保留在原关系} \rightarrow \text{作FK}\\ \text{一份剥离到新关系} \rightarrow \text{作PK}\end{cases}$
3. 示例：注意区分$\underline{\text{PK}}$，$\overline{\text{FK}}$，$\overline{\underline{\text{PFK}}}$的表示方法
   - 假设 $\begin{cases}\underline{\text{Empt}} \rightarrow \text{Ename, Dept}\\ \text{Dept} \rightarrow \text{Dname (Transitive Dependency)}\end{cases}$
   - Emp($\underline{\text{Emp}}$, Ename, Dept, Dname) → $\begin{cases}\text{Emp}(\underline{\text{Emp}}, \text{Ename}, \overline{\text{Dept}})\\ \text{Dept}(\underline{\text{Dept}}, \text{Dname})\end{cases}$

4 其他级别的NF(课外，可不看)

1. BCNF: Non-key Attribut $\xrightarrow[\times]{\text{Identify}}$ Part of Composite Key
2. 4NF:解决Multivalue Dependency
   - 多值依赖：比如若下列 $\begin{cases}\text{M} \xrightarrow{\checkmark} \text{S}\\ \text{M} \xrightarrow{\checkmark} \text{C}\\ \text{C} \xleftrightarrow{\times} \text{S}\end{cases}$ →则存在多值依赖，且记为 $\begin{cases}\text{M}\rightarrow\rightarrow\text{S}\\ \text{M}\rightarrow\rightarrow\text{C}\end{cases}$

M	S	C
M1	S1	C1
M1	S1	C2
M1	S2	C1

   - 当关系达到4NF时就认为完成Normalization了，已经没有Duplicates了

3. 5NF:解决Loss Dependency
    - 若表格拆开又拼回去不改变原表格(Lossless-Join分解)，则表格就有Loss Dependency
    - 但实际上不会如此吹毛求疵了

2. Transaction(事务)

2.1. 概念与概述

1️⃣ Tansaction定义：

1. 定义：数据库操作序列(A logical unit of work)
2. 根本特性：要么一次性执行完Entirely Completed要么Aborted，是不可分割的工作单位

状态	操作
COMMIT	事务正常结束，完成所有操作(读取数据，更新写回硬盘)
ROLL BACK	事务异常结束，中途不再继续运行，已完成的操作也全部撤回

2️⃣ Tansaction引入的目的

1. 定义Unit of Work，适用于一个/多个用户
2. Concurrent Access，允许多个Command交织执行，适用于多个用户

3️⃣ Tansaction种类

1. Implicit Transaction: 单独一句DML/DDL就是Transaction

```
SELECT / INSERT / DELETE / CREATE / ALTER / DROP -- 比如一个SELECT就是一个事务
```

2. User-Defined Transaction: 需要关键字`BEGIN/COMMIMT/RILLBACK`

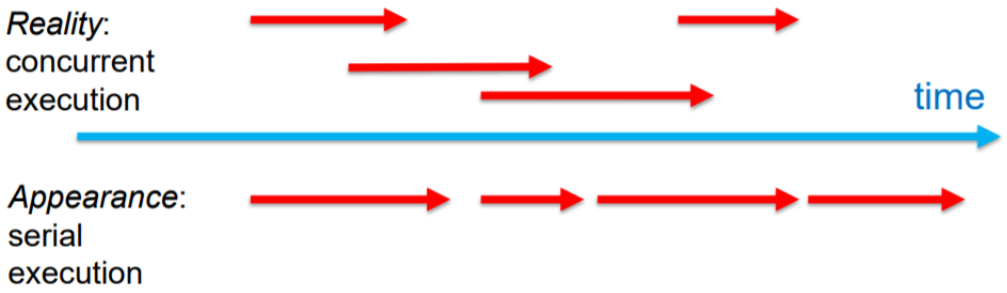
```
 -- 从SATRT到COMMIT不论中间多少语句，都只算一个Transaction
START TRANSACTION; -- 开始事务,也可以用BEGIN TRANSACTION
 -- SQL 语句1
 -- SQL 语句2
COMMIT; -- 提交事务
ROLLBACK; -- 或者如果发生错误，回滚事务
```

4️⃣ Transaction性质: ACID

性质	描述
Atomicity	Transaction是不可分割的逻辑工作单位(执行完Or回退)
Consistancy	Transaction执行前后数据库都是一致状态+多用户读取数据时应看到相同值
Isolation	运行结束前一个Transaction的改变对其他Transaction不可见
Durability	一旦Transaction执行完其对数据的改变永久有效(即使系统崩溃)

2.2. Serializability

1️⃣ Transaction的交叉执行(Interleaved)：注意Transaction $\xrightarrow{\text{缩写}}$ TXN



Execution种类	含义	备注
Concurrent/Interleaved	不同TXN交叉并行执行	实际TXN执行方式
Serial Execution	不同TXN一个个执行	最安全但最低效的执行方式

2️⃣ 示例

1. Serial Execution的读写操作

TXN	Stage1	Stage2	Stage3	Stage4
TXN1	R(A)	W(A)	\	\
TXN2	\	\	R(B)	W(B)

2. Concurrent Execution的读写操作：黄标为Concurrent Execution部分

TXN	Stage1	Stage2	Stage3	Stage4
TXN1	R(A)	R(A)	W(A)	W(A)
TXN2	\	R(B)	\	W(B)

3️⃣ Serializable

1. 含义：多个TXN是Serializable $\Longleftrightarrow$(多个TXN并行执行效果 $\equiv$ Serial Execution执行效果)
2. 上例中：若AB的读写互不干扰，则两表最终结果一样，则TXN1/2是Serializable

2.3. Concurrent Execution的问题与解决

2.3.1. Concurrent Access Conflict

😳出现问题的根本原因：多TXN操作同一Object，并且其中至少一个操作是Write

1️⃣ Loss Update Problem: 多次针对一个Object写，Object为最后一次写的内容

TXN	Stage1	Stage2	Stage3	Stage4	Stage5
Bob	R(A)	\	W(A)	\	COMMIT
Alice	\	R(A)	\	W(A)	COMMIT

- 最后结果是Alice写入内容

2️⃣ Uncommitted Data/Dirty Read Problem: 某TXN回退导致未交付数据被后面TXN使用

TXN	Stage1	Stage2	Stage3	Stage4	Stage5
Bob	R(A)	W(A)	\	\	ROLL BACK
Alice	\	\	R(A)	W(A)	COMMIT

- W(A)数据并未COMMIT但却被Alice使用

3️⃣ Inconsistent Retrieval Problem: 一个TXN在聚合函数操作+另一个TXN在Update数据

Alice	Bob
SELECT SUM(Salary) FROM Employee;	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 33;
	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 44;
(finishes calculating sum)	COMMIT;

- 理论上，可以合理安排Interleaved来避免冲突

2.3.2. Concurrency Control Method: 解决冲突

2.3.2.0. 概述

1️⃣ 总体思路：合理安排读写操作的顺序

2️⃣ 基本思路

基本思路	操作	备注
Pesimistic	先检查有无问题，以选择不执行/更正后执行	比如Lock
Optimistic	先一股脑执行，执行完后有问题再回来更正	此课程不涉及

- Optimistic方法：Timestamping和Optimistic Concurrency Control

2.3.2.1. Lock Method

1️⃣ 总体思路：

1. 逻辑上：通过让TXN持有Lock，让当前TXN霸占某个数据项(Exclusive Use of Data)
2. 物理上：在RAM上设置Lock Manager来管理Lock的数据

2️⃣ Level/Gradularity(精细度)of Lock

粒度	含义	使用频率
Database-Level	锁住整个数据库	Slightly Rare
Table-Level	锁住一张表格	Very Common
Page-Level	锁住一页(一张表存储在多页中)	Common
Row-Level	锁住表中的一行	Most Common

Gradularity	含义	使用频率
Field-Level	锁住表中的某个值	Rare

1. 理想情况Field-Level最好，但其Lock过于复杂(High Overhead)，反倒最不常用
2. 在MySQL中，如果多层级Lock被应用，则需要Intention Lock来辅助

3 Types of Lock

1. Binary Lock: 仅Lock/Unlock两种状态，只要用到数据(不论读写)都是Lock
2. Shared and Exclusive Locks(Read and Write Locks)
    - 描述

Type	适用情况	要求
Share/Read Lock	Read Table Only	无Write Lock
Exclusive/Write Lock	Update(Write) Records in Table	无其他Lock

    - 要求表，这张表很重要

操作	已有S-Lock	已有X-Lock
申请新的S-Lock	✔	✘
申请新的X-Lock	✘	✘

4 上锁与解锁Two Phase Policy

1. Two Phase Policy:

Phase	描述
Grow Phase	给Object一个个上锁
Shrink Phase	给Object一个个解锁，当解锁开始时就不能再上锁了

2. Strict Two Phase Policy: 相比于非Strict有如下改变
    - 只有在Transaction COMMIT后才能解锁
    - 解锁不是一个个解锁，而是一次性全部解锁

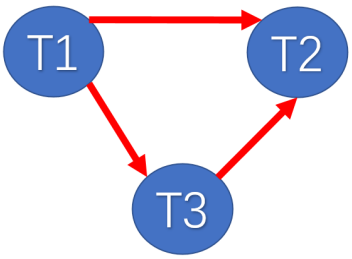
2.3.2.2. Unresolvable Problem For Lock: Deadlock

1 基本思路

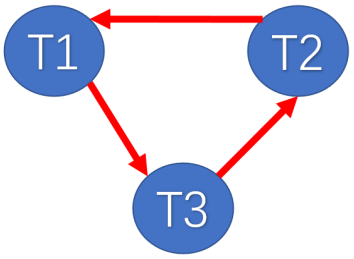
基本思路	操作	备注
Pesimistic	执行前先检查是否死锁，如果死锁就不执行/更正后执行	N/A
Optimistic	先一股脑执行，每隔一段时间检测是否锁死并加以解决	此处不涉及

2 死锁Detection的方法：Wait-For Graph

1. 用圈代表TXN，用箭头表示等待(A → B表示A在等待B释放某个Lock)
2. 当Graph构成闭环→ 死锁，但注意成环不代表闭环，闭环是要形成环路(下图示例)



闭环❌



闭环✔

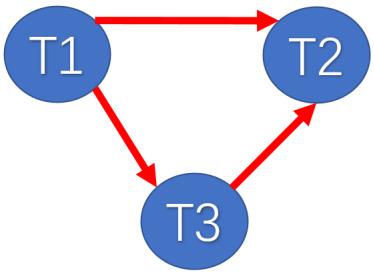
3 死锁检测例题1: 序列1如下，C=COMMIT，其次不必在意一个Object可以不读就写

序列: TXN	1	2	3	4	5	6	7	8	9
序列1: T1	R(Z)	\	\	\	W(Y)	C	\	\	\
序列1: T2	\	W(X)	W(Y)	\	\	\	C	\	\
序列1: T3	\	\	\	W(Y)	\	\	\	C	\

1. Lock请求时序：$T_{1,2,3}$对X-Lock(Y)的请求构成等待

T1	T2	T3
R(Z) $\xrightarrow{\text{获得}}$ S-Lock(Z)	\	\
\	W(X) $\xrightarrow{\text{获得}}$ X-Lock(X)	\
\	W(Y) $\xrightarrow{\text{获得}}$ X-Lock(Y)	\
\	\	W(Y) $\xrightarrow[T_2\text{释放}]{\text{等待}}$ X-Lock(Y)
W(Y) $\xrightarrow[T_{2,3}\text{释放}]{\text{等待}}$ X-Lock(Y)	\	\

2. 对于$T_{1,2,3}$的Wait-For Graph $\xrightarrow{\text{不闭环}}$不构成死锁



3. TXN顺序：T2完成$\xrightarrow[\text{T3获得X-Lock(Y)}]{T_2\text{ COMMIT释放X-Lock(Y)}}$T3完成$\xrightarrow[\text{T1获得X-Lock(Y)}]{T_3\text{ COMMIT释放X-Lock(Y)}}$T1完成

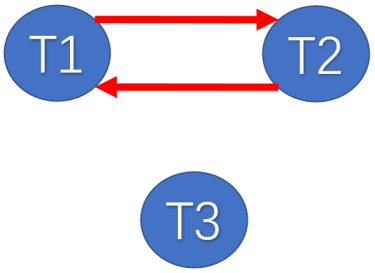
4 死锁检测例题2

序列: TXN	1	2	3	4	5	6	7	8	9
序列2: T1	\	W(Y)	\	\	\	W(X)	C	\	\
序列2: T2	W(X)	\	\	\	W(Y)	\	\	C	\
序列2: T3	\	\	R(Z)	W(Z)	\	\	\	\	C

1. Lock请求时序：$T_{1,2,3}$对X-Lock(Y)的

T1	T2	T3
\	$W(X) \xrightarrow{获得} X\text{-}Lock(X)$	\
$W(Y) \xrightarrow{获得} X\text{-}Lock(Y)$	\	\
\	\	$R(Z) \xrightarrow{获得} S\text{-}Lock(Z)$
\	\	$W(Z) \xrightarrow{获得} X\text{-}Lock(Z)$
\	$W(Y) \xrightarrow[T_1释放]{等待} X\text{-}Lock(Y)$	\
$W(X) \xrightarrow[T_2释放]{等待} X\text{-}Lock(X)$	\	\

2. 对于$T_{1,2,3}$的Wait-For Graph $\xrightarrow{闭环}$构成了死锁



3. TXN顺序：只有T_3可以完成

5 一些课外补充

1. 死锁的解决：选择一个死锁的TXN去(部分)回退，至于是哪个可以根据Age/负载等
2. 死锁的预防：Wait-die and Wound-die Policy

2.3.2.3. Concurrency Control Method

1 TXN序列：是一个Loss Update Problem问题

Time	t=0	t=1	t=2	t=3	Final
T1	R(A)	\	\	W(A)	COMMIT
T2	\	R(A)	W(A)	\	COMMIT

2 用Binary Lock解决

1. 规则：

操作	已占用Lock
申请新的Lock	✗

2. 执行

阶段	操作
T1-R(A)	获取Lock(A)并执行
T2-R(A)和T2-W(A)	等待T1释放Lock(A)
T1-W(A)	已有Lock(A)并执行
COMMIT	只有W(A)完成执行

3 Shared and Exclusive Locks(Read and Write Locks)解决

1. 规则

操作	已占用S-Lock	已占用X-Lock
申请新的S-Lock	✔	✘
申请新的X-Lock	✘	✘

2. 执行

阶段	操作
T1-R(A)	获取S-Lock(A)-1
T2-R(A)	获取S-Lock(A)-2 (在已有S-Lock情况下在获取一个S-Lock是OK的)
T2-W(A)	等待T1释放X-Lock(A)-1
T1-W(A)	等待T2释放X-Lock(A)-2
COMMIT	 ⟹ 死锁，谁都COMMIT不了

2.4. Transaction Log

1 含义：记录数据库事务变化的日志，包括

1. Transaction开始的记录
2. 每个SQL语句的详细信息
3. Transaction COMMIT的记录

2 作用：将所有更改恢复到最后一次COMMIT的状态