Transaction & Concurrency Control

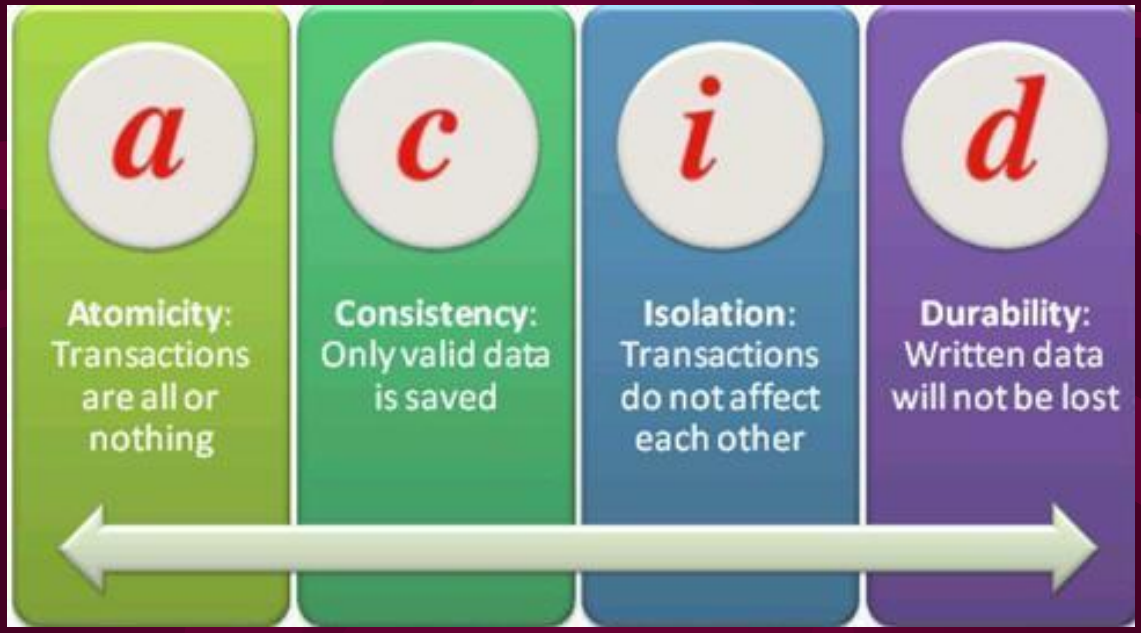
SQL Transaction and Data Control

Transaction Control Language	
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to its original values
Data Control Language	
GRANT	Gives a user permission to take a system action or access a data object
REVOKE	Removes a previously granted permission from a user

ACID Properties

- **ACID** (*Atomicity*, *Consistency*, *Isolation*, *Durability*) is a set of properties of database transactions
- In the context of databases, a single logical operation on the data is called a “transaction”
- A single transaction may involve changes to multiple database tables
- These are properties of a “reliable” DBMS, and starting in the late 1970s vendors developed technologies to achieve them automatically

Database “Acid” Properties



Transactions

- A transaction is one or more database updates that must either succeed or fail as a group; it is an indivisible logical unit of work
- For example a transaction to transfer money from one account to another must complete both the reduction in the balance of one account and the increase in the balance of the other account

- Another common example is the processing of an order:
  - Add row to Order table
  - Add multiple rows to LineItem table
  - Update rows of product table to decrease "quantity on hand"
- SQL uses the "COMMIT" command to determine the end of a transaction
- Transaction control is implemented via different types of locks (i.e. read locks or write locks) and possibly with logging of changes to the database

The Transaction Logs

- Keeps track of all transactions that update the database
- DBMS uses the information stored in a log for:
  - Recovery requirement triggered by a ROLLBACK statement
  - Program's abnormal termination
  - System failures (hardware, power, etc.)
- Some RDBMS's may have one log or a separate before and after log

Before Image

- The before image log keeps a copy of every database row before it was changed by a transaction, including the transaction id, type of change, and timestamp
- This log is used for rollback (undo) of transactions
- The log may also contain indexing (or physical) pointers to thread transactions
- This log may also be used for “read consistency” services

After Image

- The after image log keeps a copy of every database row after it was changed by a transaction, including the transaction id, type of change, and timestamp
- This log used for rollforward (redo)
- Checkpoints are noted in the log
- The log may also contain indexing (or physical) pointers to thread transactions
- This log may also be used for replication services

Checkpoint

- For maximum efficiency, database changes are not written to disk as they occur
- Database is paged into memory, and pages are written back to disk only as necessary
- A checkpoint synchronizes the disk copy(s) and memory (paged) copy of a database; all database activity is paused for a checkpoint
- The disk copy is typically mirrored (or RAID used) to minimize the effect of disk hardware problems

Media (or other platform) Failure (rollforward)

- To rollforward (or redo transactions), “after images” since the last checkpoint are applied to the database

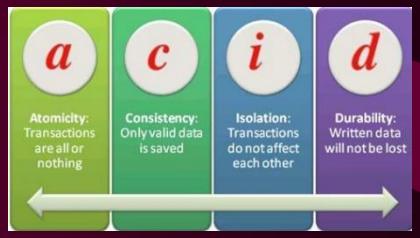
Application Failure (Rollback)

- To rollback (or undo one or more transactions), each changed row is replaced by its before image version

Lost Update Problem (The account should have $650)

Instruction	T1's Balance	T2's Balance	Database Balance
Status before the update			500
T1 reads the account record	500		500
T2 reads the account record	500	500	500
T1 adds $50	550	500	500
T2 adds $100	550	600	500
T1 updates the account record	550	600	550
T2 updates the account record	550	600	600

Serializable



- Most modern multi-user databases can guarantee "Serializable" (isolated) transactions
- That is the *effect* of a transaction T1 will be preserved and not be seen by another transaction T2 until T1 has successfully completed
- In effect all concurrent transactions will product the same result as if they had all run one at a time in a serial manner

Concurrency Control

- Coordination of the simultaneous transactions execution in a multiuser database system
  - Objective: ensures serializability of transactions in a multiuser database environment
- Important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems
  - Three main problems are lost updates, uncommitted data, and inconsistent retrievals

Concurrency Control (con't)

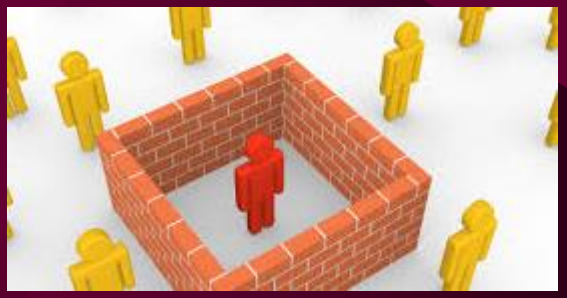
- Serializability is one method describing types and levels of concurrency control - maintaining database integrity with concurrent users
- Concurrency control is implemented differently in different RDBMS's usually via locking, timestamping, versioning, and/or version control numbering, but the goal of each is to allow for parallel activity without compromising the integrity of the database

Isolation Level

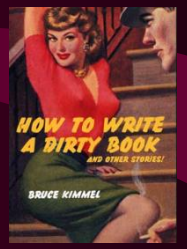
- The SET TRANSACTION SQL command can set an isolation level
- There are several levels (varies by vendor):
  - SERIALIZABLE (normally the default, exclusive locking until after commit) – reads/locks on all involved rows of tables involved in a transaction
  - READ UNCOMMITTED (no locks, “dirty read”, can read uncommitted rows)
  - READ COMMITTED (only read committed rows, non exclusive lock [others can read or process])
  - READ REPEATABLE (shared locking, others can read but not update) - reads on all rows of a result set are repeatable

Isolation Level (con't)

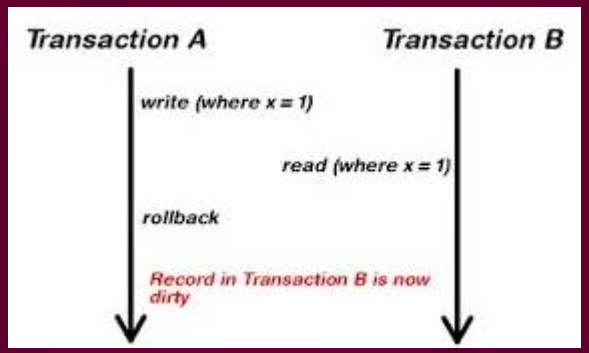
- If any level other than the first is chosen (perhaps for efficiency reasons), then the application needs to take care of concurrency control itself *via exclusive (write) lock statements* on any data retrieved for updating purposes



Dirty Read

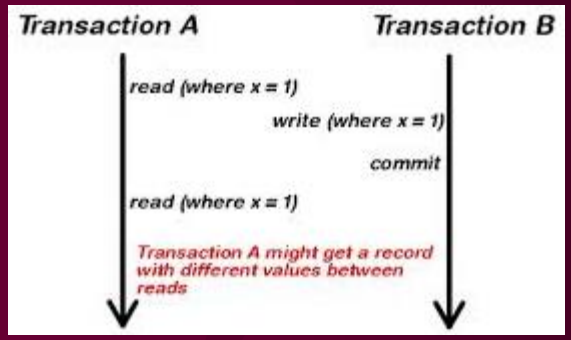


- Suppose transaction T1 performs an update on some row, transaction T2 then retrieves that row, and transaction T1 then terminates for some reason (rollback)
- Transaction T2 has seen a row that no longer exists, and in a sense that never existed



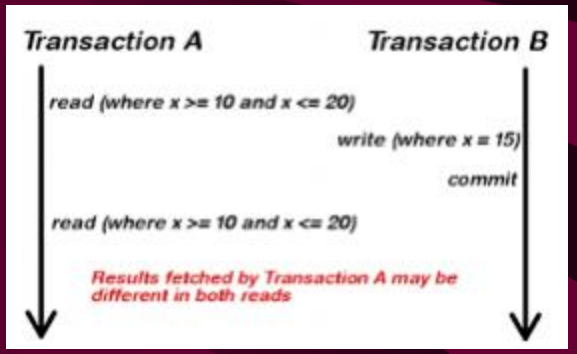
Nonrepeatable Read

- Suppose transaction T1 retrieves a row, transaction T2 then updates that row, and then transaction T1 retrieves that same row again
- Transaction T1 has now retrieved the same row twice but seen different version



Phantoms

- Suppose transaction T1 retrieves the set of all rows that satisfy some condition (Price > 50)
- Suppose that T2 then inserts a new row satisfying the same condition
- If T1 now (as part of the same transaction) repeats its retrieval requests, it will see a row that did not previously exists



Violations of Serializability

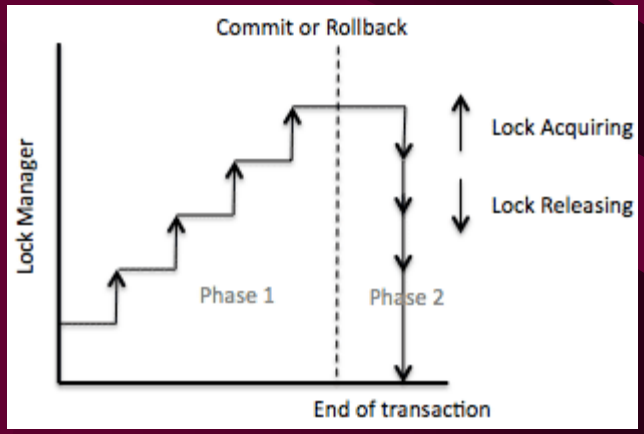
Isolation Level	dirty read	nonrepeatable read	phantoms
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

COMMIT/ROLLBACK (Implicit Locking)

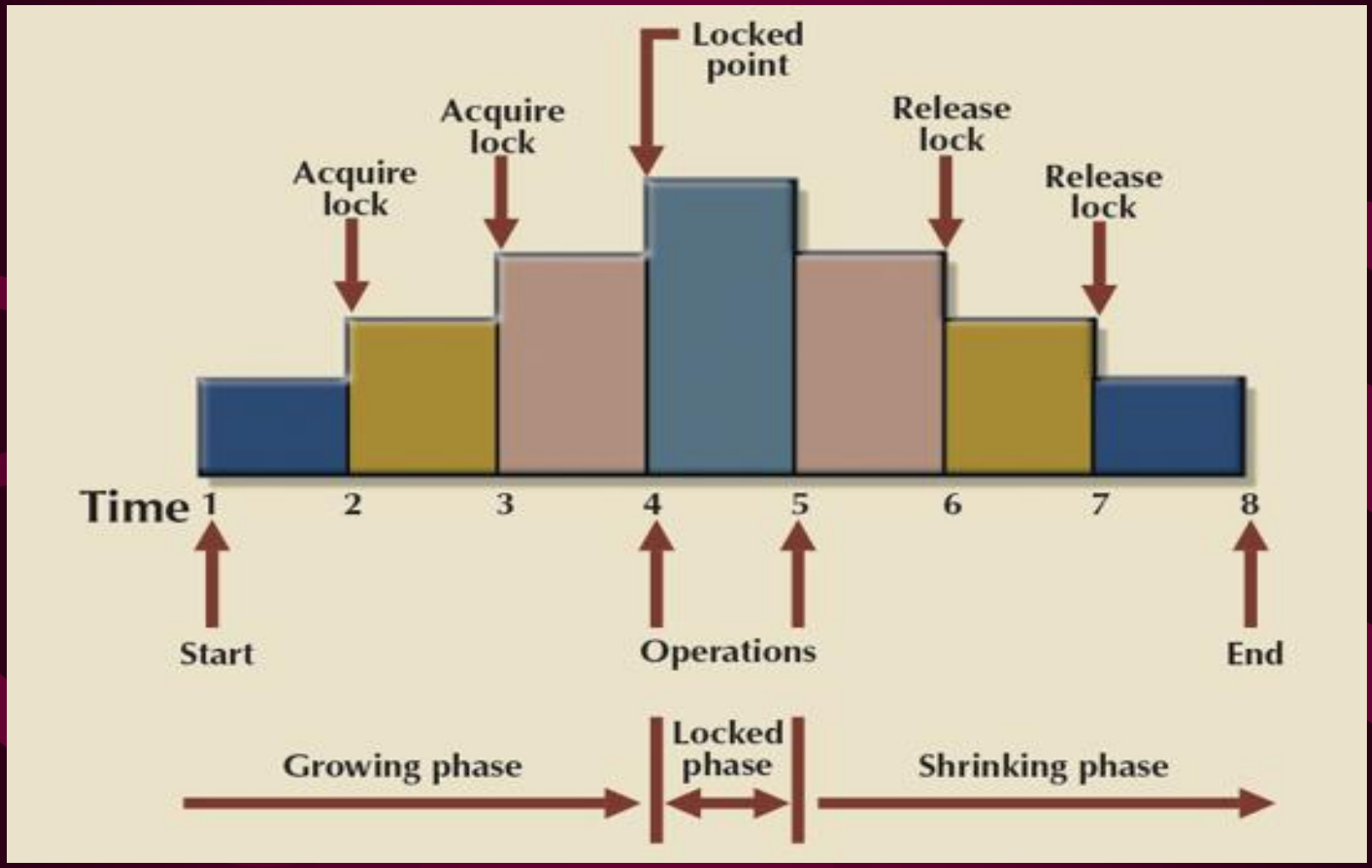
- Implicit database locking is accomplished by using Serializable isolation level with use of COMMIT and ROLLBACK
- Issue COMMIT command if all parts of transaction are successful
- Issue ROLLBACK command if all parts are not successful

Two Phase Locking

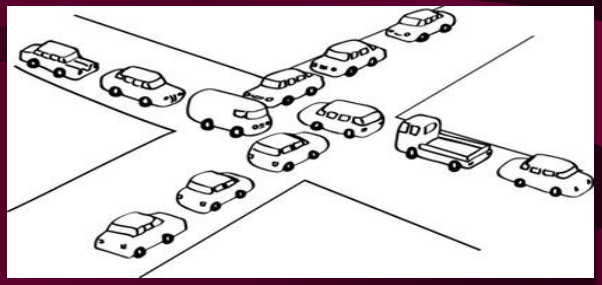
- Most RDMS use *two phase locking* on transactions; transactions acquire locks (write/exclusive locks for rows to be updated, and shared/read locks for rows to be read), but once first lock is relinquished, no others are obtained



Two Phase Locking (con't)



Deadlock



- Two (or more) users are waiting on access to rows (or tables) that the other has locked
  - Suppose that T1 and T2 are both to update row A and row B
  - T1 locks A, while T2 locks B
  - Then T1 tries to lock B and goes into a wait state; T2 tries to lock A and goes into a wait state
- Sometimes called "deadly embrace"
- RDBMS have various schemes for detecting and removing deadlock typically by canceling one transaction

Lock Granularity

- Some RDBMS's can lock at the table, row, and field levels
- Many RDMS cannot lock at the row level, only:
  - Database level, table level, or page level (part of an OS file)
- Makes deadlock situations more frequent
- Makes for slower response times
- Resolving lock conflicts:
  - table subdivision
  - key division

Database Level Locking

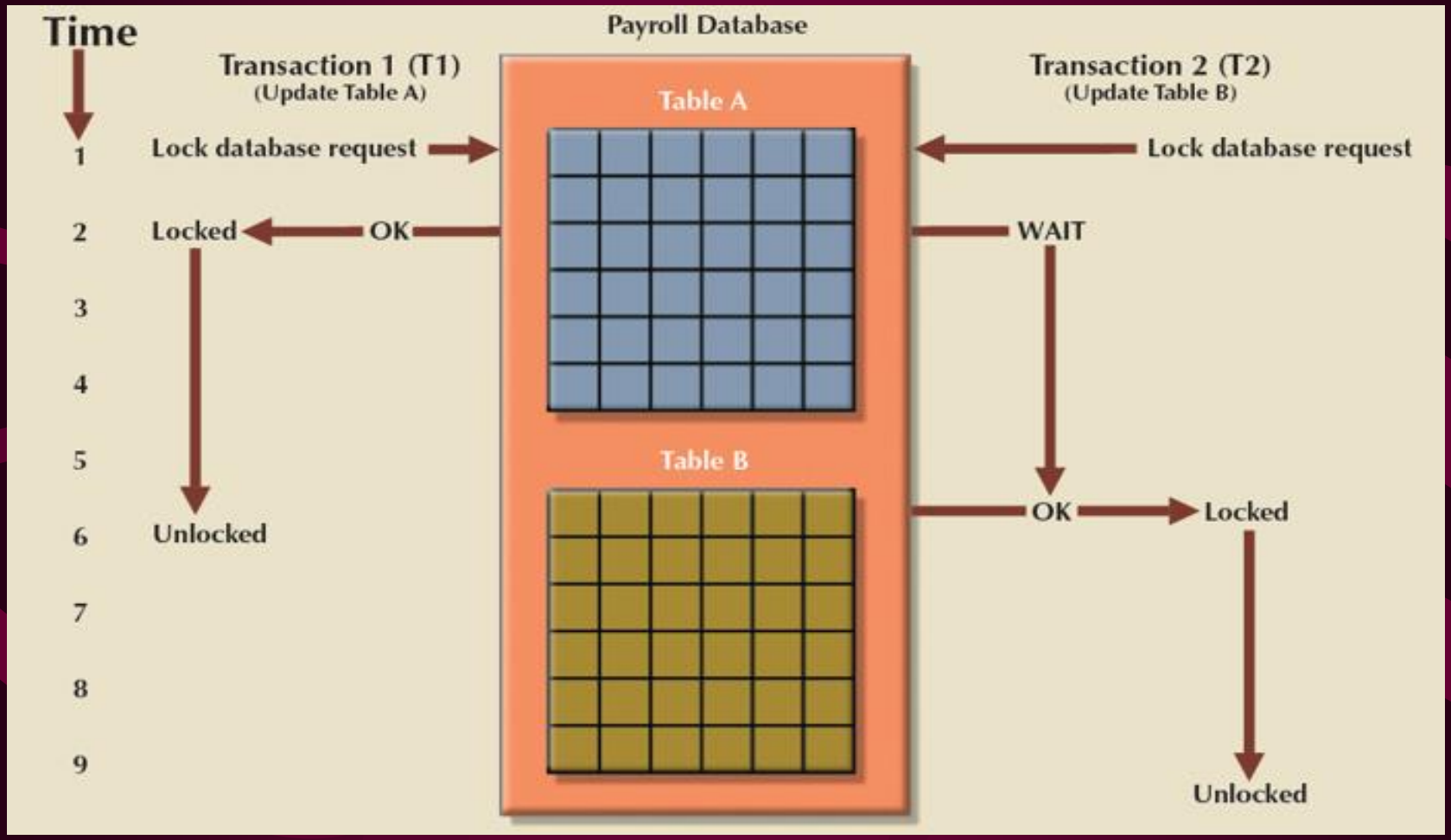
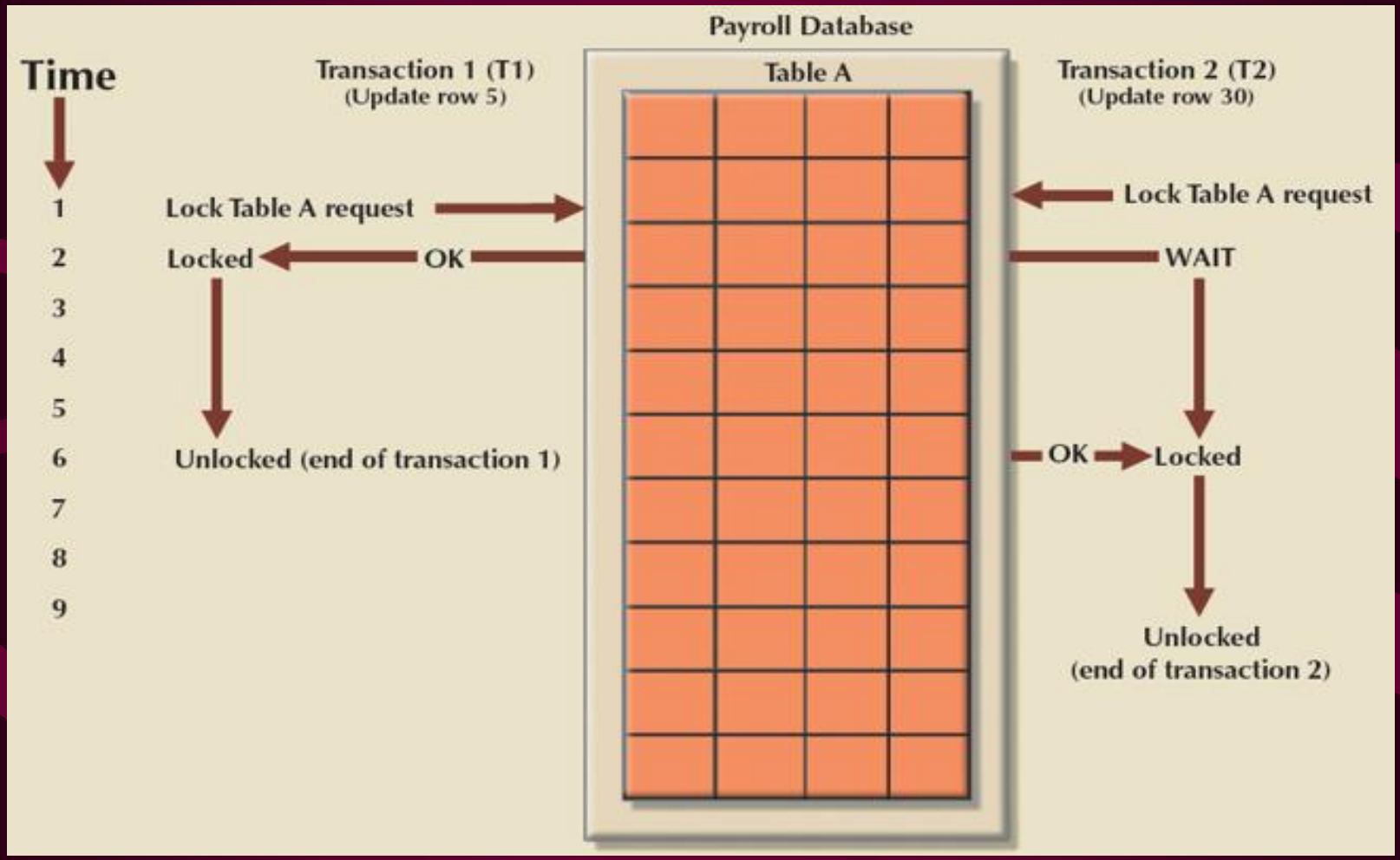
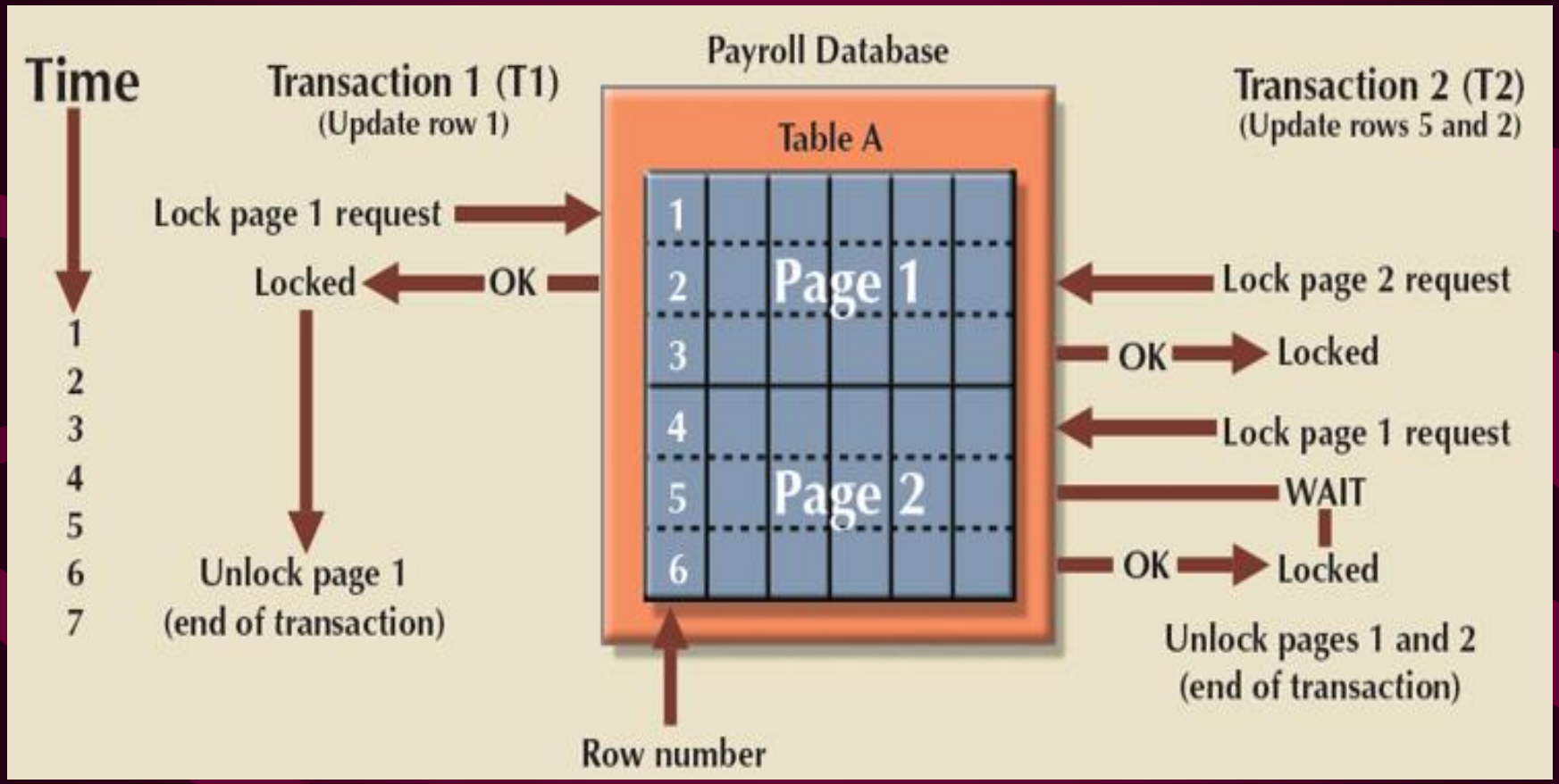


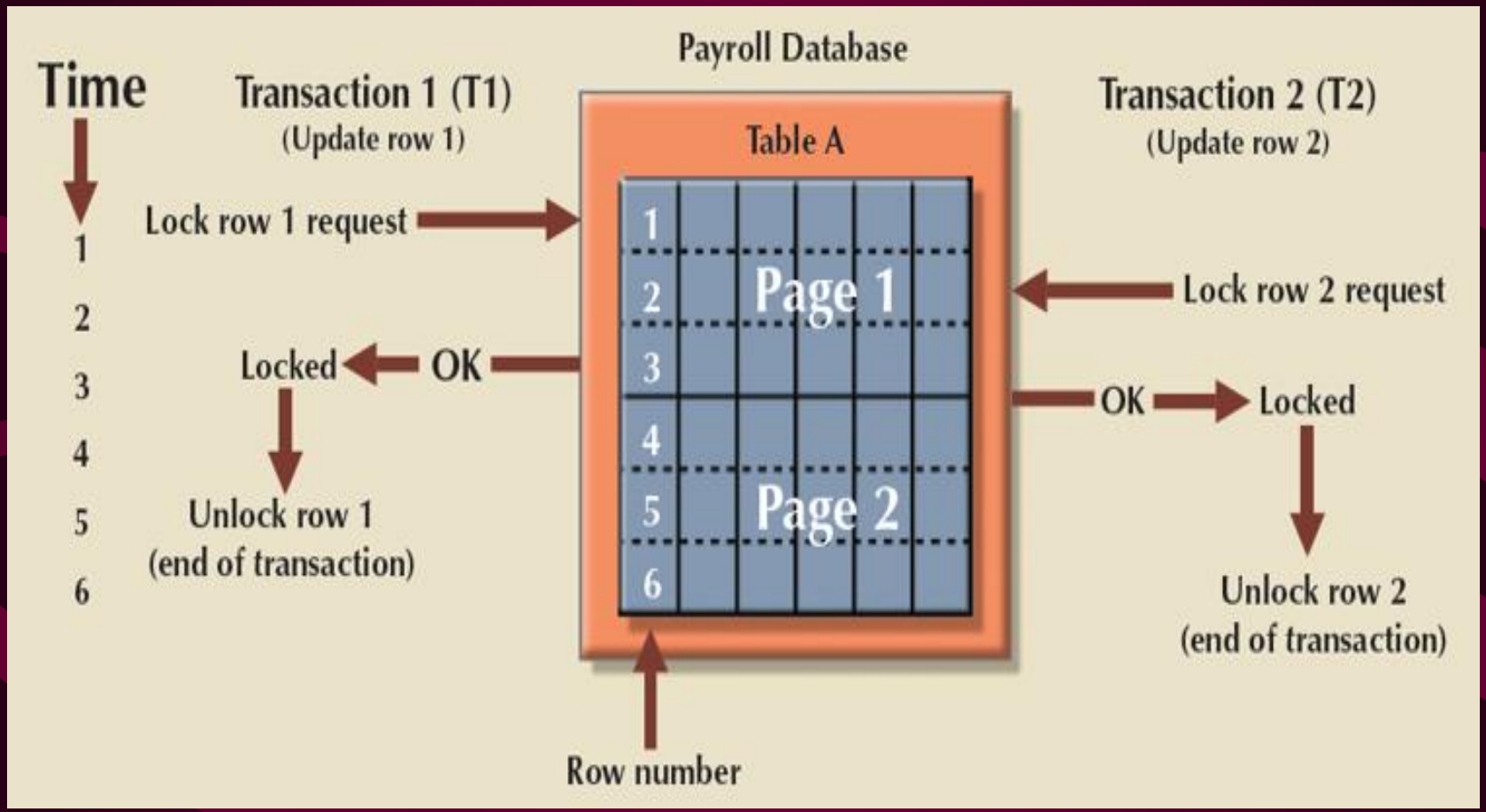
Table Level Locking



Page Level Locking



Row Level Locking



Lock Types

- Binary lock
  - Two states: locked (1) and unlocked (0)
    - If an object is locked by a transaction, no other transaction can use that object
    - If an object is unlocked, any transaction can lock the object for its use
- Exclusive lock
  - Access is reserved for the transaction that locked the object
- Shared lock
  - Concurrent transactions are granted read access on the basis of a common lock

Other Concurrency Mechanisms

- Timestamps - check last update timestamp to see if any other transaction updated row
- Before/After comparisons - check all fields (or just an update # field) ["optimistic control"]
- Normally used for interactive situations, so that a terminal (or client) operator cannot leave a screen in an idle mode with rows locked

Snapshots

- Snapshots offer a mechanism for read consistency without table locks
- Two methods:
  - Physical snapshot
  - Using rollback logs (described earlier)
- For a report, the RDBMS reads each row and checks to see (in the rollback logs) if it had been modified since the time (or system commit number) of the run start

Choosing Isolation Level

- For most reports, uncommitted reads are ok; if the report had to be fully accurate as of a point in time, then committed read level would be appropriate
- For maintenance purposes, where a single row is to be updated, then repeatable read is appropriate so that other users cannot update the same record at the same time
- For transactions involving multiple tables and rows, full isolation level is appropriate

Concurrency Control with Optimistic Methods

- Optimistic approach: based on the assumption that the majority of database operations do not conflict
  - Does not require locking or time stamping techniques
  - Transaction is executed without restrictions until it is committed
  - Usually implement in the application program rather than the RDBMS
- Phases of optimistic approach for multiple table updates:
  - Read
  - Validation
  - Write

Optimistic Methods (con't)

- Read phase
  - Transaction:
    - Reads the database
    - Executes the needed computations
    - Makes the updates to a private copy of the database values
- Validation phase
  - Transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database
- Write phase
  - Changes are permanently applied to the database

Optimistic Methods (con't)

- Single table optimistic method
  1. Read record (including update counter)
  2. Make changes in temporary space
  3. Re-read record to see if update counter the same
  4. If counter the same, write record with incremented update counter
  5. If counter not the same, go back to step 1

Update Options

- Database updating options (CACHE on/off)
  - Deferred-write technique or deferred (cache) update
    - Transaction operations do not immediately update the physical database
    - Only transaction log is updated
  - Write-through technique or immediate update
    - Database is immediately updated by transaction operations during transaction's execution

Access & MySql

- Some products may not directly offer transaction control and/or concurrency control
- Only recently did Access include a simple type of concurrency control, but just for some simple forms created via their wizards
- With MySql, you can choose to use tables with or without transaction control; transaction control requires more resources (InnoDB)
- You might best handle some concurrency control in your application programs via before/after comparisons

MySQL ISAM vs InnoDB

- **Data integrity and foreign key constraints**
  Foreign keys establish a relationship between columns in one table and those in another. For example, you might create a library application where books can be loaned to members. A foreign key constraint would ensure that a member existed before a book could be checked-out. Similarly, removing a user would not be possible until all their books were returned.
- **2. Transactions**
  InnoDB tables support transactions. A transaction allows multiple SQL commands to be treated as a single and reliable unit.
  - Consider a banking application where you are transferring money from one account to another. The transaction would only be committed if both accounts were altered successfully. If anything failed, the database would be rolled-back to a previous state.
  - In addition, InnoDB tables recover well from crashes. MySQL will analyze the log files to ensure the data is accurate so there is no need to repair tables.
- **3. Row-level locking**
  InnoDB uses row-level rather than table-level locking. If a row is being inserted, updated or deleted, only changes to the same row are held up until that request has completed. Tables that receive more updates than selects may be faster with InnoDB.
- Creating an InnoDB table is no more complex than MyISAM, e.g.
- CREATE TABLE employee (id smallint(5) unsigned NOT NULL, firstname varchar(30), lastname varchar(30), PRIMARY KEY (id)) ENGINE=InnoDB; However, designing that database with foreign key relationships does require more effort. Database novices will find MyISAM easier because it has fewer features.
- **No full-text search**
  InnoDB tables do not support full-text searches; it is not easy to match one or more keywords against multiple columns.
- **Slower performance**
  If your application is primarily selecting data and performance is a priority, MyISAM tables will normally be faster and use fewer system resources.

References

- Transaction Processing: Management of the Logical Database and its Underlying Physical Structure (Data-Centric Systems and Applications) by Seppo Sippu and Eljas Soisalon-Soininen
- Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery (The Morgan Kaufmann Series in Data Management Systems) by Gerhard Weikum and Gottfried Vossen
- Concurrency Control and Recovery in Database Systems by Philip Bernstein, Vassos Hadzilacos, et al

Homework

- Textbook Chapter 10
- Review Questions 1 thru 13