Exam Structure:

1. Yes or No (20 Marks)
    a. Correct +2
    b. Wrong -1
2. DB Design - ER & Relational DB (20 Marks)
3. Relational Algebra & Functional Dependency & Normal Forms (30 Marks)
4. Transaction & Indexes (20 Marks)
5. Spatial Data using Relational Algebra (10 Marks)

Topics not covered:

- *PLpgSQL*
- *SQL*
- *Multi-versioning*
- *Optimistic Concurrency Control*
- *Graph DB (Week 10 & 11)*

LECTURES TO OUTLINE:

1. 1-4, 7-16

Intro
- DBMS Concepts
- Other Concepts

DB Design
- Conceptual DB Design
    - ER Model
    - Constraints on Relationship Types
    - Attributes of Relationship Types
    - Representing the ER Model
    - Entity-Relationship Diagrams (ERDs) & Occurrence Diagrams
    - ERD Common Notation
- Enhanced ER (EER) Model
    - Specialisation

Relational Data Model
- Structure
    - RD Model vs. ER Model
    - Keys
- Integrity Constraints
    - Checking Constraints on Updates
- ER to Relational Data Model Mapping

Relational Algebra

Functional Dependency
- Anomalies
- Functional Dependencies and Armstrong's Axioms
- Closure
- Candidate Key

Normal Forms
- Normal Forms
- Relational DB Design

- Disks and Files
  - Disks
    - Disk Space Management
    - Buffer Management
    - Buffer Replacement Policies
  - Records & Blocks
    - Fixed-Length
    - Variable-Length
  - Files
- File Organisation and Index
  - File Organisation
  - Indexes
    - Clustered vs Unclustered Index
    - Dense vs Sparse Indexes
    - Primary vs Secondary Indexes
- Transaction Management
  - Transaction Problems
  - Recovering from Failures
  - Concurrency Control
  - Concurrency Control Methods
- Spatial DB

Intro

DBMS Concepts

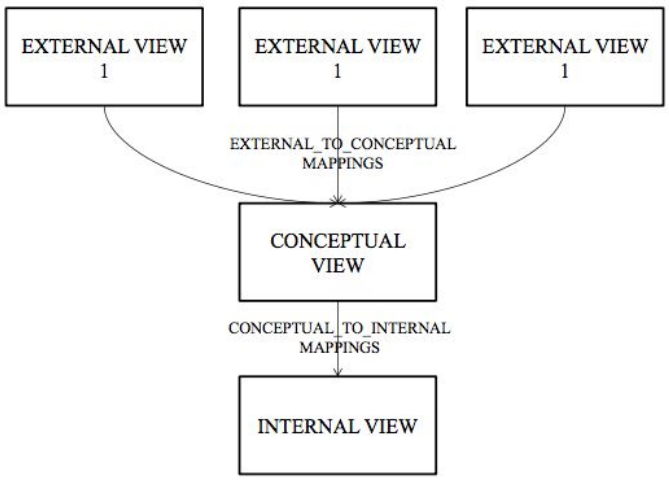
- Data Model
  - A set of concepts that is used to describe the allowed structure of a DB (i.e. the metadata structure)
- Database Schema
  - An instance of a data model, that is, a description of the structure of a particular DB in the formalism of the data model
- Database Instance
  - The data in the DB at a particular time

Process:

1. Define a DB be specifying its schema
2. The state is then an empty instance of the schema
3. To create the initial instance we load in data
4. After this, each change in state is an update

Other Concepts

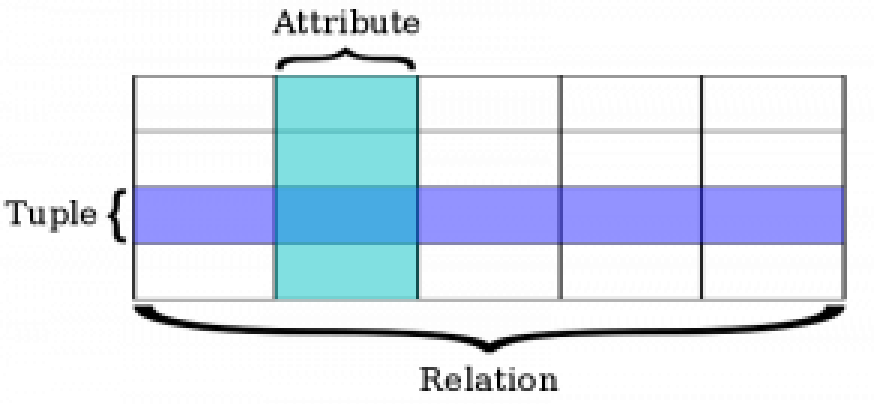
1. Metadata
   a. A definition and description of the stored database, such as structure of each file, type and storage format of each data item, constraints, etc.
2. ANSI-SPARC Three Level Architecture
   a. The *external* or *view level* includes a number of external schemas or user views
   b. *Conceptual level* has a conceptual schema describing structure of DB for users
   c. *Internal level* has an internal schema describing physical storage structure of DB



   d. In the three level architecture:
      i. Data definition language (DDL) - used to define the conceptual schema
      ii. View definition language (VDL) - used to define external schemas
      iii. Storage definition language (SDL) - used to define the internal schemas
      iv. Data manipulation language (DML) - used to construct retrieval requests (queries) and update requests
   e. Abstraction
      i. Three levels of abstraction
      ii. Two levels of data independence
         1. Logical data independence
            a. The ability to change the conceptual schema without changing external views
            b. Does change the external-to-conceptual mapping
         2. Physical data independence

        a. The ability to change physical storage paths and access structures without changing the conceptual view
    f. Does change the conceptual-to-internal mapping

3. DB Components
    a. Runtime DB Processor - Receives retrieval and update requests and carries them out with the help of the stored data manager
    b. Stored Data Manager (File Manager) - Controls access to the DBMS information stored on disk
    c. Pre-Compiler - Extracts DML commands from the host language program
    d. Query Processor - Parses high-level queries and converts them into calls to be executed by the data manager



DB Design

Conceptual DB Design

ER Model

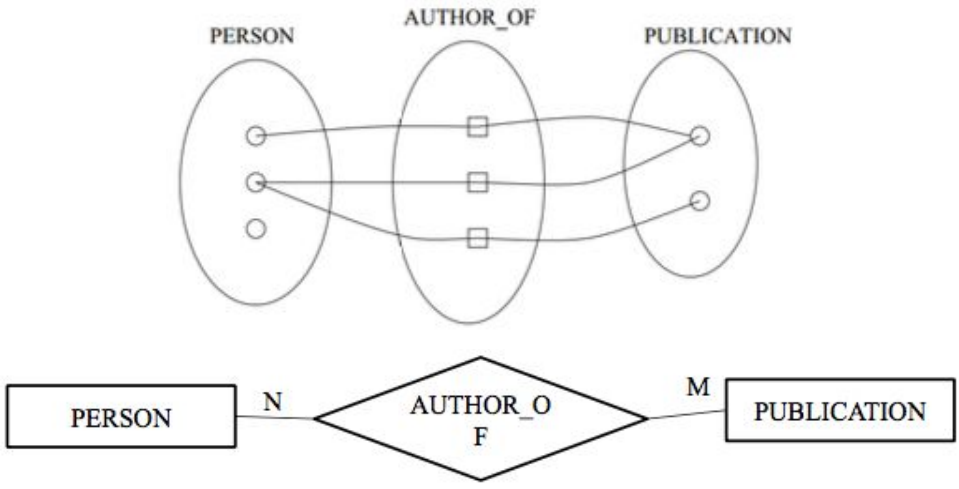
This is a high level conceptual data model which is used mainly as a design tool

- **Entity type**
  - Group of objects (entities) with the same properties (attributes)
  - Identified by a **key**
    - Set of attributes that uniquely identifies an entity (e.g. {name} is a key of dept)
    - **Key Constraint**
      - In any extension of the entity type, there cannot be two entities having the same values for their key attributes
  - The set of individual entity instances at a point in time is an **extension** of the entity type
- **Entity**
  - Member of an entity (object) type
- **Attribute**
  - Describe a property of entities
  - Simple (sex = 'female') or composite (e.g. name(title, initials, family name))
    - Each simple attribute has a value set (domain) which is the set of possible values that the attribute can take
    - A composite attribute has multiple domains for each attribute type
  - Single-valued (e.g. student number) or multi-valued (e.g. {Colour})
  - Derived Attribute - it's value can be derived from other attributes (e.g. # of students)
- **Relationship** (among objects)
  - Represents an association between things
    - A relationship type (R) among (n) entity types (E1,...,En) is a set of associations among entities from these types
      - "E1,...,En participate in R"
    - Each instance r = (e1,...,en) in R is a relationship

Constraints on Relationship Types

Relationships can be constrained by certain factors in the real world

- E.g. A research grants supports only one research project, but a research project can be supported by many grants
  - PROJECT:GRANT is a 1:N relationship
- E.g. an N:M relationship could be the authorship of publications



Participation Constraint:

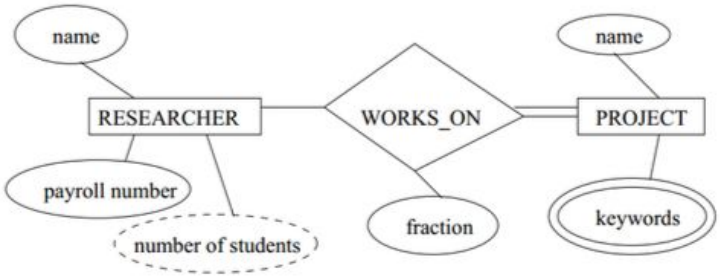
- Participation of an entity in a relationship can be:

- Total - every entity must participate
- Partial - not necessarily total (e.g. below)

Attributes of Relationship Types

Relationship types can have attributes

- E.g. a researcher may work on several projects using FRACTIONs of time

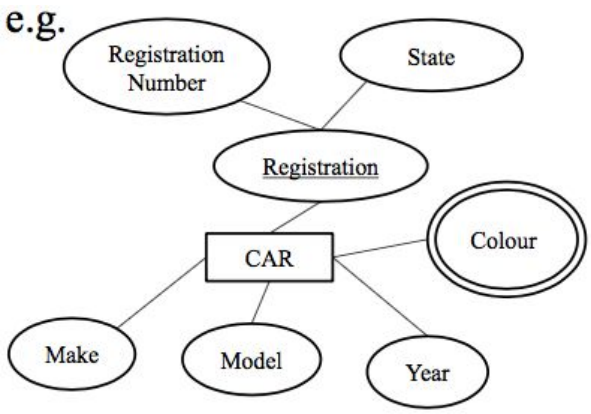


Representing the ER Model

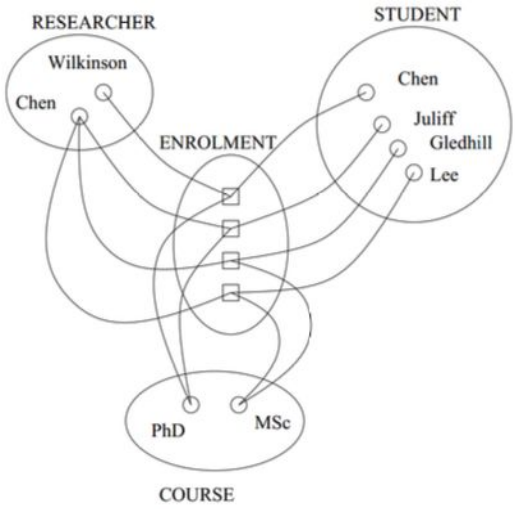
- Composite attributes - ()
- Multi-valued attributes - {}

CAR Registration(Registration No, State), Make, Model, Year, {Colour}
((ARQ) 595, Vic), Datsun, 120Y, 1972, {green}) ((8HR) 696, WA), Mazda, 929, 1979, {grey, black})

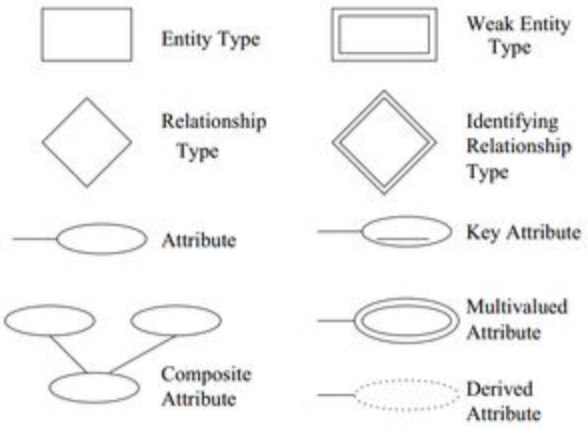
Entity-Relationship Diagrams (ERDs) & Occurrence Diagrams



&



ERD Common Notation



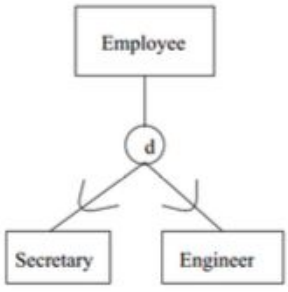
Enhanced ER (EER) Model

Used to represent the requirements from applications as accurately and explicitly as possible

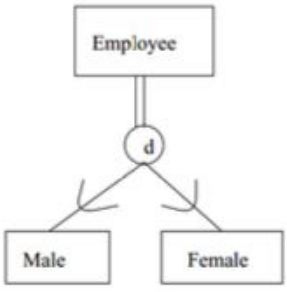
Specialisation

The process of defining a set of subclasses of an entity type

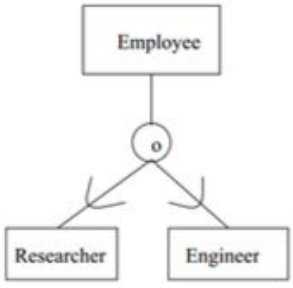
- A subclass inherits all the attributes of the superclasses
- Process:
    - Define a set of subclasses of an entity type
    - Associate additional specific attributes with each subclass
    - Establish additional specific relationship types between each subclass and other entity types, or other subclasses
- A subclass may have multiple superclasses
- Can be total or partial
- Can be disjoint or overlapping



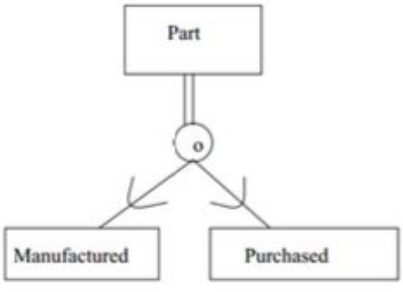
Partial disjoint



Total disjoint



Partial Overlapping



Total Ovelapping

Relational Data Model

In the relational model, everything is described using relations. It has been implemented in most commercial database systems because it uses a simple and uniform data structure.

Structure

- Each column of a table corresponds to a named attribute
    - The set of allowed values for an attribute is called its domain
- Each row of the table is called a tuple of the relation
    - A tuple $t(A_1,A_2,...,A_n)$ is a point in domain $dom(A_1) \times \ldots \times dom(A_n)$
    - A relation (or a relation instance) is a set of tuples: a subset of $dom(A_1) \times \ldots \times dom(A_n)$
    - A relation schema is used to describe a relation
    - The degree of a relation is the number of attributes of its relation schema
- There is no ordering of the column or rows
- Composite and multivalued attributes are not allowed in this structure

PLAYER					
Name	Position	Goals	Age	Height	Weight
Heady	Half-forward	17	24	183	83
Sumich	Full-forward	59	26	191	92
Langdon	Utility	23	23	189	86

RD Model vs. ER Model

- Relation schema (intension) ⟷ Entity or relationship type schema (intension)
- Attributes ⟷ Attributes
- Tuple ⟷ Instance of entity/relationship
- Relation (instance, extension) ⟷ Entity/relationship extension
- Composite and multivalued attributes are allowed only in the ER model

Keys

Keys are used to identify tuples in a relation

- A superkey is a set of attributes that uniquely determines a tuple
- A candidate key is a superkey, none of whose proper subsets is a superkey
- A primary key is a designated candidate key
    - It is often necessary to invent a primary key (e.g. Person_ID)

Integrity Constraints

- Key Constraint
    - Candidate key values must be unique for every relation instance
- Entity Integrity
    - An attribute that is part of a primary key cannot be NULL
- Referential Integrity
    - Relates to ‘foreign keys’ which refer to a tuple in another relation, in which the value of FK must occur in the other relation or be entirely NULL

Checking Constraints on Updates

To maintain the integrity of the database, we need to check that constraints are not violated before updating

ER to Relational Data Model Mapping

One technique for DB design is to first design a conceptual schema using a high-level data model, and then map it to a conceptual schema in the DBMS data model for the chosen DBMS

For ER → RD Model there are 7 steps:

1. Entities
    a. For each regular (not weak) entity type E, create a relation R with:
        i. Attributes (all simple attributes, and *, of E)
        ii. Key (choose one of the keys of E as the primary key for the relation
    b. For each specialised entity type E, with parent entity type P, create a relation R with:
        i. Attributes (the attributes of the key P, plus the simple attributes of E)
        ii. Key (the key of P)
2. Weak Entities
    a. For each weak entity type W, with owner entity type E, create a relation R with:
        i. Attributes (all simple attributes of W, and *, include as a FK the prime attributes of the relation derived from E
        ii. Key (the foreign key plus the partial key of W)
3. 1:1 Relationships
    a. For each 1:1 relationship type B, let E and F be the participating entity types, let S and T be the corresponding relations
        i. Choose one of S and T (prefer one that participates totally), say S
        ii. Add the attributes of the primary key of T to S as a FK
        iii. Add the simple attributes (*) of B as attributes of S
4. 1:N Relationships
    a. For each 1:N relationship type B, let E (1) and F (N) be the participating entity types, let S and T be the corresponding relations
        i. Add the attributes of the primary key of S to T as a foreign key
        ii. Add to T any simple attributes (*) of the relationship
5. N:M Relationships
    a. For each N:M relationship type B, create a new relation R. Let E and F be the participating entity types, let S and T be the corresponding relations
        i. Attributes (the key of S and the key of T as FK, plus the simple attributes of B)
        ii. Key (the key of S and the key of T)
6. Multivalued attribute A
    a. For each multivalued attribute A, create a new relation R. Let A be an attribute of E
        i. Attributes (1: simple attribute), 2: composite attribute)
            1. A together with the key of E as a FK
            2. The simple components of A together with the key of E as a FK
        ii. Key (all attributes)
7. N-ary Relationship Type
    a. For each n-ary relationship type (n>2) create a new relation with:
        i. Attribute (the key of S and and the key of T as FK, plus the simple attributes of B)
        ii. Key (the key of S and the key of T, except that if one of the participating entity types has participation ratio 1, it's key can be used as a key for the new relation

*simple components of composite attributes

Relational Algebra

See notebook

Functional Dependency

Anomalies

A good database schema should not lead to *update anomalies*

- Redundancy in a DB means storing a piece of data more than once
    - Often useful for efficiency
    - Creates potential for consistency problems
- A poor redundancy control may cause update anomalies
    - **Modification Anomaly**
        - If data is stored in multiple places, it needs changing in ALL places, or inconsistency occurs
    - **Insertion Anomaly**
        - If data is entered into a new area, it may be inconsistent with prior data
    - **Deletion Anomaly**
        - If the last data item in a relation is deleted, the association between the entities is lost

Functional Dependencies and Armstrong's Axioms

See notebook

Closure

See notebook

Candidate Key

See notebook

Normal Forms

Criteria for a good database design (i.e. to resolve update anomalies)
A relation scheme can be in several different normal forms

Normal Forms

1. 1NF
    a. Attribute values are atomic - meaning no multivalued attributes or composite attributes
        i. Often results in a large amount of duplicate entries for certain attributes
2. 2NF
    a. If all non-prime attributes are fully functionally dependent on the relation keys
        i. A prime attribute is one that is part of a candidate key
3. 3NF
    a. If for all non-trivial FD's of the form $X \rightarrow A$ that hold, either X is a superkey or A is a prime attribute

    ALTERNATIVE DEFINITION:
    b. If every nonprime attribute is fully functionally dependent on the keys and not transitively dependent on any key
4. Boyce-Codd NF (BCNF)
    a. If whenever $X \rightarrow A$ holds, and $X \rightarrow A$ is non-trivial, X is a superkey
5. 4NF (Multivalued Dependencies)
6. 5NF (Join Dependencies)

Relational DB Design

Anomalies can be removed from relation designs by decomposing them until they are in a normal form
A good decomposition should have the following two properties:

1. Dependency Preserving
2. Lossless Join

Disks and Files

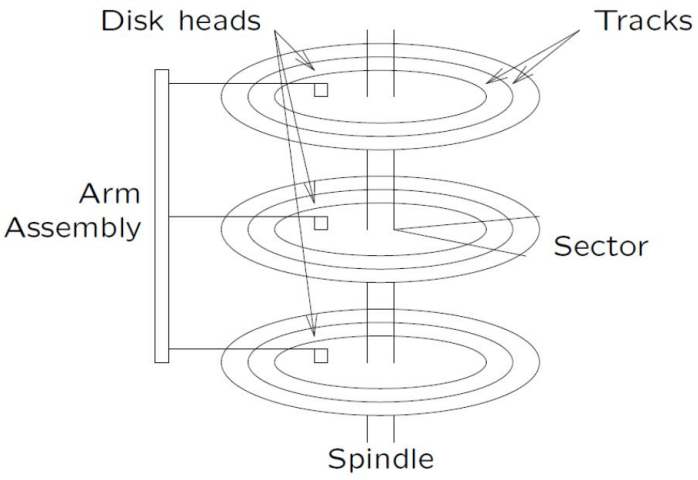
Storage types:

- Primary (main memory)
    - Fast access, expensive
- Secondary (hard disk)
    - Slower access, less expensive
- Tertiary storage (tapes, cd, etc)
    - Slowest access, cheapest

Disks

Disk characteristics:

- Made up of:
    - Platters > tracks > sectors (blocks)
- Transfer unit: 1 block
- Access via block address
- If a single record in a block is needed, the entire block is transferred



Access Time:

- Seek time (find the right track ~10 msec)
- Rotational delay (find the right sector ~5 msec)
- Transfer time (read/write block ~10 usec)

Disk Space Management

Disk space is managed by the disk space manager:

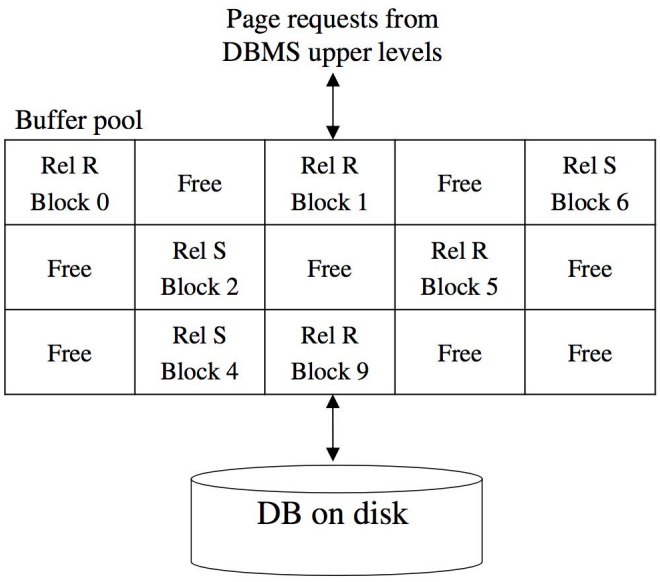
1. Improving disk access
    a. Using knowledge of data access patterns we can:
        i. **Cluster** - put two records often accessed together in the same block
        ii. Sequential - put two records scanned sequentially on the same track
2. Keeping track of free blocks
    a. Maintain a list of free blocks
3. Using OS File System to manage disk space

Buffer Management

Manages traffic between disk and memory by maintaining a **buffer pool** in main memory

Buffer pool:

- Is a collection of *page slots* (frames) which can be filled with copies of disk block data



Operations:

1. **Request_block** (replaces read_block)
    a. If block is in buffer pool:
        i. Use the copy from the buffer (unless write-locked)
    b. If not in the buffer pool:
        i. Choose a frame for replacement using replacement policy
        ii. If frame chosen is dirty, write block to disk
        iii. Read requested page into now-vacant buffer frame
            1. Dirty = false
            2. Pin Count = 0
        iv. Pin count +1 for the frame containing requested block
        v. Return address of frame containing requested block
2. **Release_block**
    a. Indicates that block is no longer in use - good candidate for removal
3. **Write_block**
    a. Updates contents of page in pool and set dirty bit on
4. **Force_block**
    a. Commits by writing to disk

Considerations:

- For each frame we need to know:
    a. Is it currently in use
    b. Has it been modified since loading (**dirty bit**)
    c. How many transactions are currently using it (**pin count**)
    d. Optional - time-stamp for most recent access

Buffer Replacement Policies

Possible schemes:

1. LRU - Least Recently Used
2. FIFO - First In First Out
    a. Have to maintain a queue of frames
    b. Add to tail when read in
3. MRU - Most Recently Used
4. Random

Records & Blocks

Records are stored within fixed-length blocks

A block (page) is:

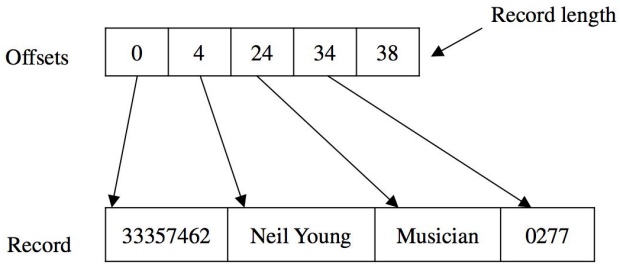
- A collection of *slots* → where each slot contains a record
- A record is identified by its ID = <page ID, slot number>

Fixed-Length

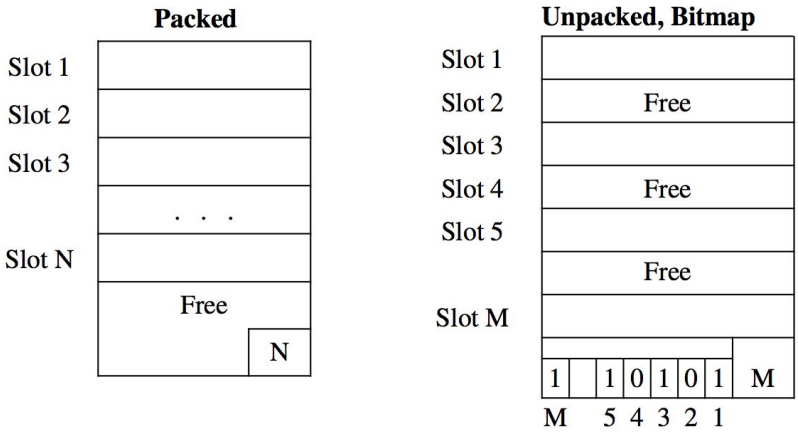
Each field has a fixed length as well as the number of fields

- Useful for intra-block space management
- Possibly wastes a lot of space

• length + offsets stored in header



- For fixed-length records, we use record slots



Variable-Length

Some field is of variable length

- Complicates intra-block space management
- Does not waste (as much) space

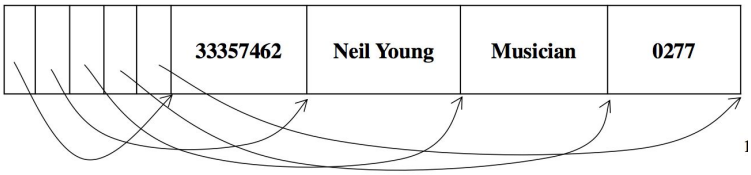
• Prefix each field by length

4 xxxx 10 Neil Young 8 Musician 4 xxxx

• Terminate fields by delimiter

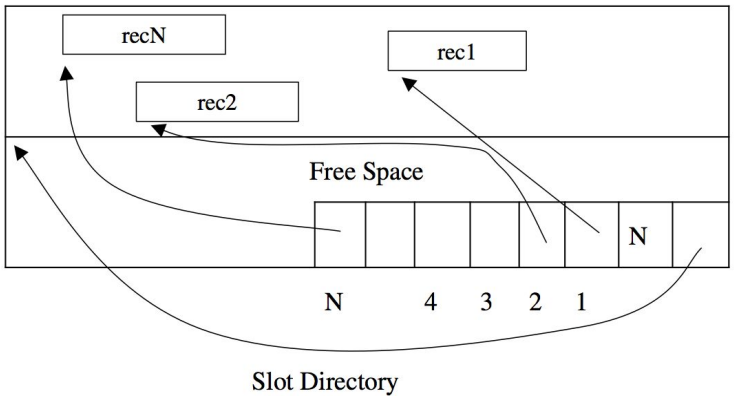
33357462/Neil Young/Musician/0277/

• Array of offsets

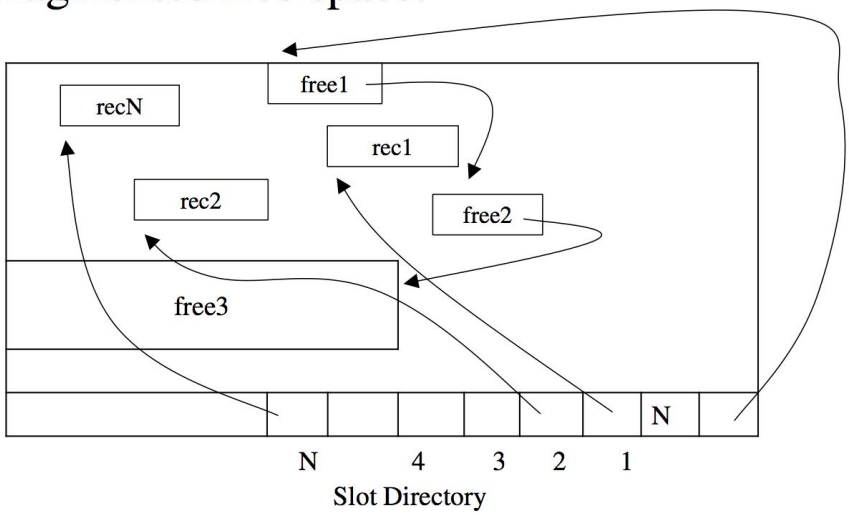


- For variable-length records - use slot directory
    - Handling free-space within block (often use a combination of these)
        - Compacted (one region of free space) - cheap to maintain
        - Fragmented (distributed free space) - space saving

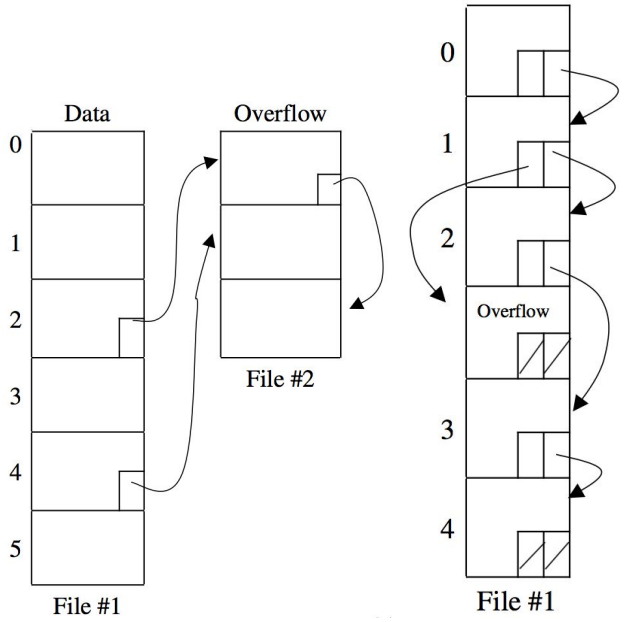
Compacted free space:



Fragmented free space:



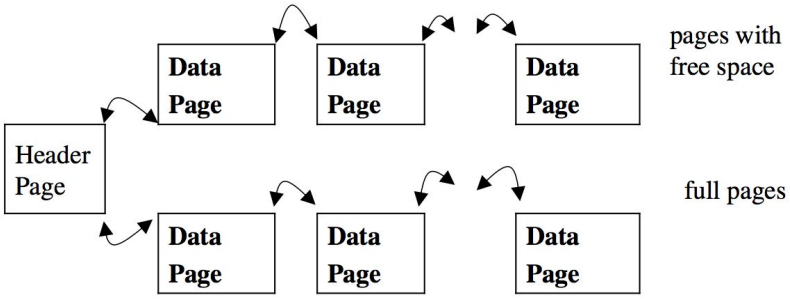
- Variable-length records sometimes have “overflow” between blocks when a block is full



Files

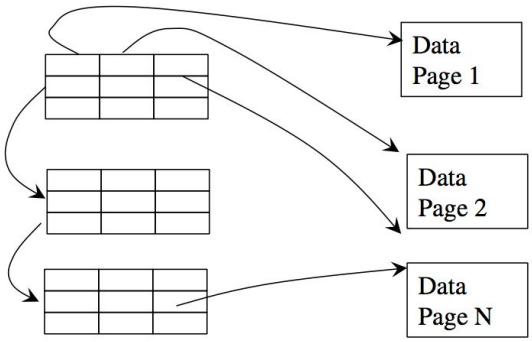
A file consists of several data blocks. Heap Files are unordered pages (blocks)
To maintain the block information:

- Linked List of pages
    - Disadvantage
        - All pages will be on the free list of records if variable length
        - To insert a record, several pages may be retrieved and examined



Organized by a Linked List

- Directory of pages
    - Each directory entry identifies a page (or sequence of pages)
    - Each entry maintains a bit to indicate if the corresponding page has any free space



File Organisation and Index

File Organisation

3 File Organisations:

1. **Heap Files**
2. **Sorted Files** - sorted on a search key - a combination of one or more fields
3. **Hashed Files** - pages in a file are grouped into “buckets” defined by a hash function

Computing Costs of each File Type:

- D - average time to read or write a disk page
- C - average time to process a record
- H - time required to apply a hash function to a record

Operations to be investigated:

1. Scan - fetch all records in a file
2. SWES (Search with equality selection) - find all records with a = ?
3. SWRS (Search with range selection) - find all records a after x
4. Insert - insert a given record into the file
5. Delete - delete a record with a given record ID

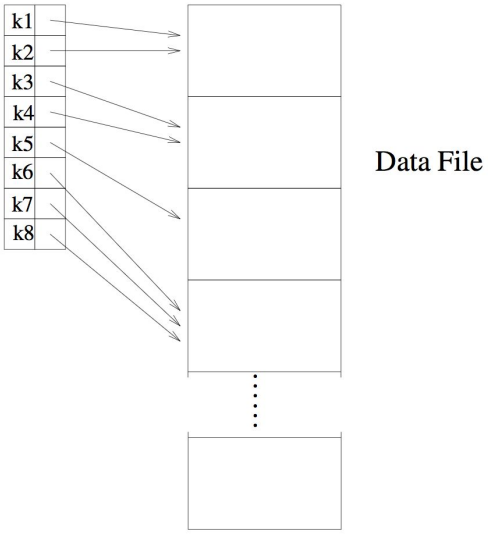
File Type	Scan	Equality Search	Range Search	Insert	Delete
Heap	BD	0.5 BD	BD	Search + D	Search + D
Sorted	B D	D log B	D log B + # matches	Search + BD	Search + BD
Hashed	1.25 BD	D	1.25 BD	2 D	Search + BD

Note:
B = number of pages
R = average number of records in a page (block)

Indexes

Think book index:

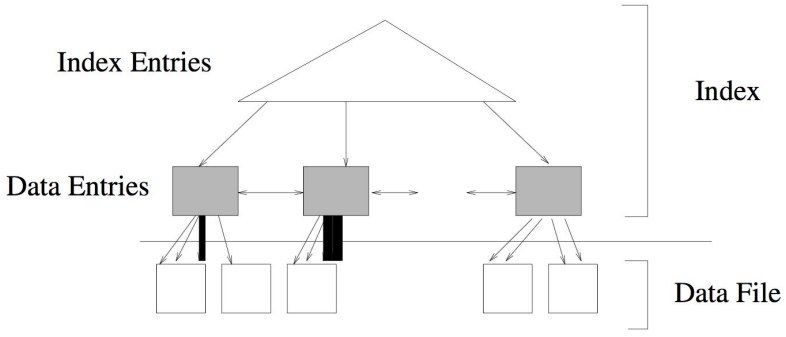
- A table of key values k*, where each entry gives places where the key is used
- Each data entry contains enough info to retrieve (one or more) records with search key value k



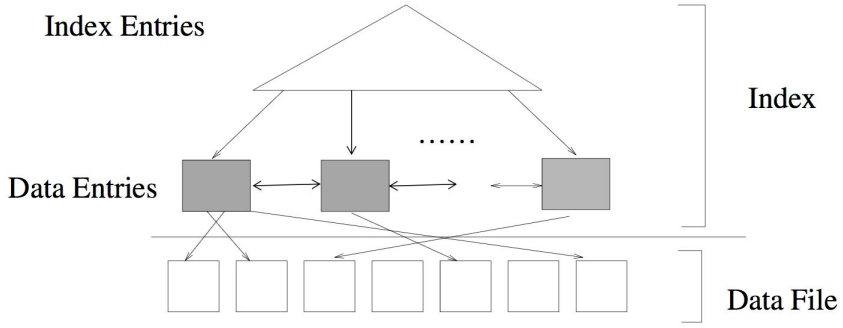
- Alternatives for data entries, k*:
  - Actual data record
  - (k, rID) pair
  - (k, rID - list) pair, where rID-list is the list of record IDs of data records with search value k

Clustered vs Unclustered Index

Clustered Index: A file is organised of data records in a similar or same way to the ordering of data entries in some index



Unclustered index



Clustered vs Unclustered:

- Clustered are relatively expensive to maintain and can only be clustered on at most one search key

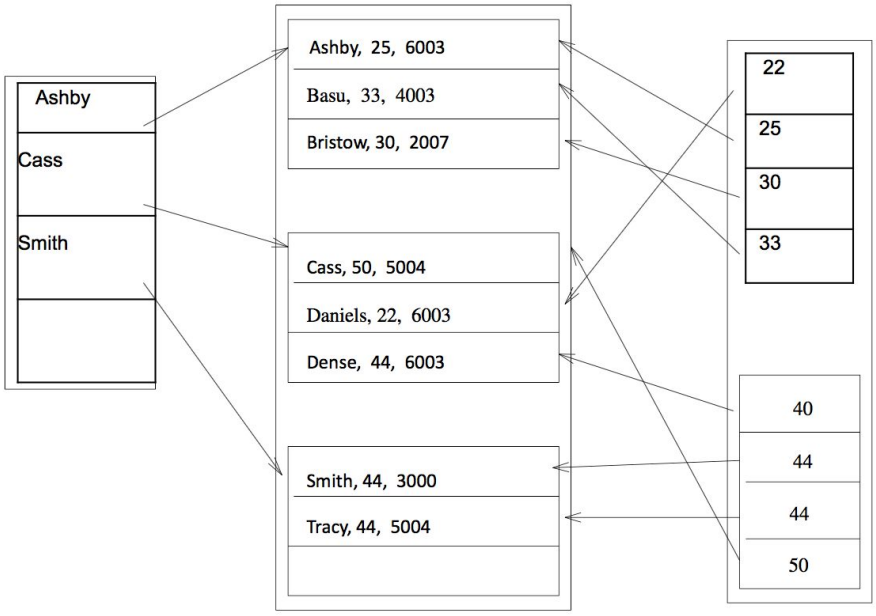
Dense vs Sparse Indexes

Dense (RHS)

- Contains (at least) one data entry for every search key value

Sparse (LHS)

- Index records are created for only some of the records



Primary vs Secondary Indexes

Primary:

- Indexing fields include primary key
- Guaranteed to not include duplicates

Secondary

- Not primary
- May include duplicates

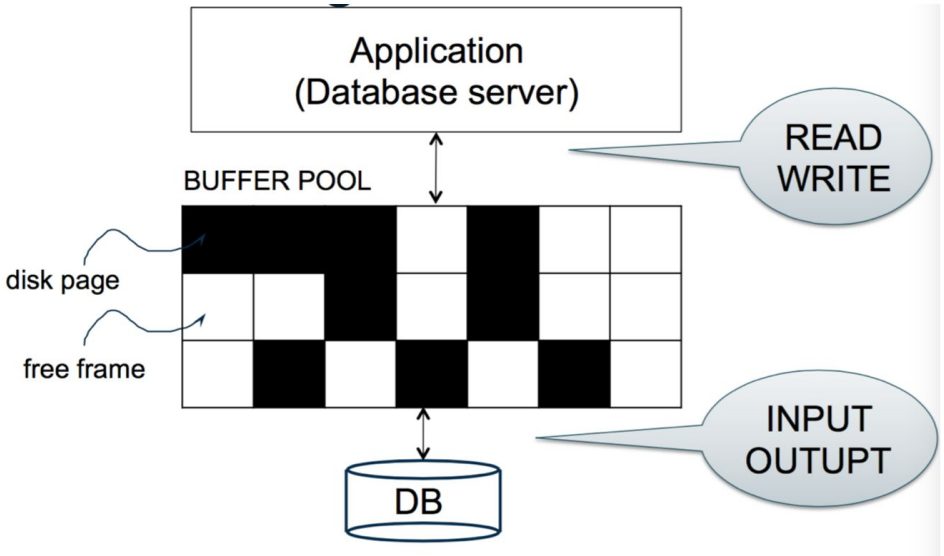
Transaction Management

A transaction is treated as an **atomic unit**
Program unit:

- A transaction that either:
    - Accesses the contents of the database
    - Changes the state of the database

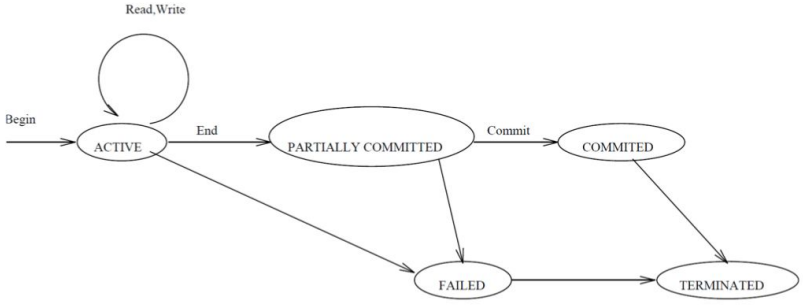
Operations:

1. Read
    a. Compute the data block that contains the item to be read, by either:
        i. Finding a buffer containing the block
        ii. Reading from disk into a buffer
    b. Copy the value from the buffer
2. Write
    a. Compute the disk block containing the item to be written, by either:
        i. Finding a buffer containing the block
        ii. Reading from disk into a buffer
    b. Copy the new value into the buffer
    c. At some point - write the buffer back to disk
3. Computation



Processing States:

1. Partially Committed Point
    a. Check and enforce correctness of the concurrent execution
2. Committed State
    a. Execution is concluded successfully



Desirable Properties of Transaction Processing - **ACID**

1. **Atomicity**
   a. A transaction is performed only in its entirety
2. **Consistency Preserving**
   a. A correct execution of the transaction must take the DB from one consistent state to another
3. **Isolation**
   a. A transaction should not make its updates visible to other transactions until it is committed
4. **Durability**
   a. Once a transaction is committed, the DB changes made must never be lost due to subsequent failure

Transaction Problems

Lost Update Problem

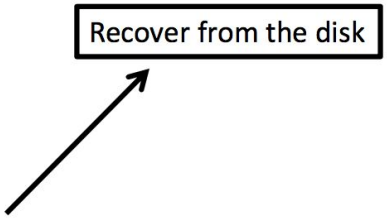
- Isolation is not enforced - update is lost
- Example below: T1's X ← X+N is lost in the DB (DB should have X = 113)

Suppose initially that $X = 100$; $Y = 50$; $N = 5$ and $M = 8$.

Database	T_1	T_2
$X = 100, Y = 50$	$X =?, Y =?$	$X =?$
$X = 100, Y = 50$	read(X) $X = 100, Y =?$	$X =?$
$X = 100, Y = 50$	$X \leftarrow X + N$ $X = 105, Y =?$	$X =?$
$X = 100, Y = 50$	$X = 105, Y =?$	read(X) $X = 100$
$X = 100, Y = 50$	$X = 105, Y =?$	$X \leftarrow X + M$ $X = 108$
$X = 105, Y = 50$	write(X) $X = 105, Y =?$	$X = 108$
$X = 105, Y = 50$	read(Y) $X = 105, Y = 50$	$X = 108$
$X = 108, Y = 50$	$X = 105, Y = 50$	write(X) $X = 108$
$X = 108, Y = 50$	$Y \leftarrow Y - N$ $X = 105, Y = 45$	$X = 108$
$X = 108, Y = 45$	write(Y) $X = 105, Y = 45$	$X = 108$

Temporary Update Problem

Database	T_1	T_2
$X = 100, Y = 50$	$X =?, Y =?$	$X =?$
$X = 100, Y = 50$	read(X) $X = 100, Y =?$	$X =?$
$X = 100, Y = 50$	$X \leftarrow X + N$ $X = 105, Y =?$	$X =?$
$X = 105, Y = 50$	write(X) $X = 105, Y =?$	$X =?$
$X = 105, Y = 50$	**FAILS**	read(X) $X = 105$
$X = 105, Y = 50$		$X \leftarrow X + M$ $X = 113$



3 possible cases:

1. Only half of T1 has been executed
2. Only half of T1 has been executed
3. T2 has been lost

Database	T_1	T_2
$X = 105, Y = 50$		$X = 113$
DBMS undoes T_1		
$X = 100, Y = 50$		$X = 113$
$X = 113, Y = 50$		write(X) $X = 113$

Database	T_1	T_2
$X = 105, Y = 50$		$X = 113$
DBMS does nothing about T_1		
$X = 105, Y = 50$		$X = 113$
$X = 113, Y = 50$		write(X) $X = 113$

Database	T_1	T_2
$X = 105, Y = 50$		$X = 113$
$X = 113, Y = 50$		write(X) $X = 113$
DBMS undoes T_1		
$X = 100, Y = 50$		$X = 113$

Incorrect Summary Problem

T_1	T_3
	$sum \leftarrow 0$
	read(A)
	$sum \leftarrow sum + A$
	⋮
read(X)	
$X \leftarrow X - N$	
write(X)	
	⋮
	read(X)
	$sum \leftarrow sum + X$
	read(Y)
	$sum \leftarrow sum + Y$
	⋮
read(Y)	
$Y \leftarrow Y + N$	
write(Y)	
	⋮

In this example, the sum calculated by T3 will be wrong by N

Recovering from Failures

Log-based recovery:

1. Undo logging
2. Redo logging
3. Undo/Redo logging

System Log

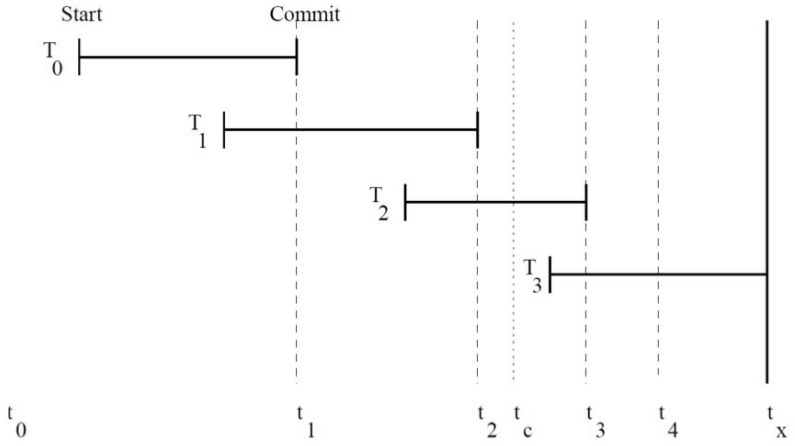
- The system records the state's information so that it can recover failures correctly
- This info is maintained in a log (aka journal or audit trail)
- Log is kept in hard disk but maintains its current contents in main memory

  - Start transaction marker [start transaction, *T*]: Records that transaction *T* has started execution.
  - [read item, *T*, *X*]: Records that transaction *T* has read the value of database item *X*.
  - [write item, *T*, *X*, old value, new value]: Records that *T* has changed the value of database item *X* from old value to new value.
  - Commit transaction marker [commit, *T*]: Records that transaction *T* has completed successfully, and arms that its effect can be committed (recorded permanently) to the database.
  - [abort, *T*]: Records that transaction *T* has been aborted.

- These entries allow the recovery manager to **rollback** an unsuccessful transaction

Recovery

- Allows us to see how the log might be used to recover from a system crash
- The database on disk will in a state somewhere between t0 and tx



- **Write-ahead Log Strategy**
  - Old data values must be force-written to the log before any change can be made to the DB
  - The transaction is regarded as committed when the new data values and the commit marker have been force-written to the log
  - t1, t2, t3 in the example above are force-written
- We can undo T1, T2, and/or T3 or backup to T0

Checkpoints - tc

- The longer between crashes, the longer recovery may take
- To avoid this problem - the system takes *checkpoints* at regular intervals
- To do this:
  - A start of checkpoint marker is written to the log
  - The DB updates in buffers are force-written
  - An end of checkpoint marker is written to the log
- In the example, on recovery - we only need redo T2

Concurrency Control

Avoiding unwanted interference between concurrent transactions is done by the **Scheduler**

Transaction Schedule

Used to optimise CPU

- Serial Schedule
  - Linear and not interleaved are assumed to be correct
- Equivalent Schedule
  - The effect on a DB of executing the first schedule is identical to the effect of executing the second
- Serialisable Schedule
  - A schedule is serialisable if it always produces the same result as some serial schedule
- Conflict Serialisable Schedules
  - Two schedules are **conflict equivalent** if:
    - Involve the same actions of the same transactions
    - Every pair of conflicting actions is ordered the same way
  - Schedule S is **conflict serialisable** if S is conflict equivalent to some serial schedule
  - Any conflict serialisable schedule is also a **serialisable schedule**
    - **The inverse is not true**

Check Conflict Serialisability

Algorithm:

1. Construct a schedule (or precedence) graph - a directed graph
2. Check if the graph is cyclic
    a. Cyclic: non-serialisable
    b. Acyclic: serialisable

Construction the graph:

1. Each transaction is a VERTEX
2. An EDGE is added from Ti → Tj if:
    a. There are two conflicting operations O1 from Ti and O2 from Tj
        i. O1 and O2 are conflicting if:
            1. They are in different transactions but on the same data item
            2. One of them must be a write
    b. O1 is before O2

OR

1. **Draw a node for each transaction in the schedule**
2. **If transaction A writes to an attribute which transaction B has read from, draw an line pointing from B to A**
    a. **aW after bR, b → a**
3. **If transaction A writes to an attribute which transaction B has written to, draw a line pointing from B to A**
    a. **aW after bW, b → a**
4. **If transaction A reads from an attribute which transaction B has written to, draw a line pointing from B to A**
    a. **aR after bW, b → a**

Concurrency Control Methods

1. **Locking Mechanism**
    a. The idea of locking some data item X is to:
        i. Give a transaction exclusive use of the data item X
        ii. Not restrict the access of other data items
    b. Prevents one transaction from changing a data item currently being used in another transaction

T_1	T_2
read_lock(Y)	
read(Y)	
unlock(Y)	
	read_lock(X)
	read(X)
	unlock(X)
	write_lock(Y)
	read(Y)
	$Y \leftarrow X + Y$
	write(Y)
	unlock(Y)
write_lock(X)	
read(X)	
$X \leftarrow X + Y$	
write(X)	
unlock(X)	

2. **Two Phase Locking**
    a. To guarantee serialisability, transactions must obey the two-phase locking protocol:
        i. Growing phase: all locks for a transaction must be obtained before any locks are released
        ii. Shrinking phase: gradually release all locks (once a lock is released no new locks may be requested)

T_1
read_lock(Y)
read(Y)
write_lock(X)
unlock(Y)
read(X)
$X \leftarrow X + Y$
write(X)
unlock(X)

3. **Deadlock Problem**
    a. A problem that arises with locking is deadlock
    b. Deadlock occurs when two transactions are each waiting for a lock on an item held by the other
    c. Deadlock Check
        i. Create a vertex for each transaction
        ii. Create an edge from A to B if A is waiting for an item locked by B
        iii. If the graph has a cycle, then a *deadlock* has occurred
    d. Deadlock Detection
        i. Periodically check for deadlocks, abort and rollback some transactions (restart them later)
    e. Deadlock Prevention
        i. Assign priorities based on timestamps:
            1. Wait-Die:
                a. If A has higher priority, A waits for B - otherwise A aborts
            2. Wound-wait:
                a. If A has higher priority, B aborts - otherwise A waits

T_1	T_2
write_lock(X)	
read(X)	
	write_lock(Y)
	read(Y)
write_lock(Y)	
waiting for Y*	write_lock(X)
waiting for Y*	***waiting for X***

4. **Timestamp Ordering**
    a. The idea is:
        i. To assign each transaction a timestamp
        ii. To ensure that the schedule used is equivalent to executing the transactions in timestamp order
    b. Each data item X is assigned:
        i. Read TS(X) - read timestamp - the latest timestamp of a transaction that read X
        ii. Write TS(X) - write timestamp - the latest timestamp of a transaction that write X
    c. Used in read and write operations in **IF statements**

Spatial DB

A spatial DB is a DBMS that is optimised to store and query basic spatial objects:

- Simple
    - Point (location)
    - Line (road, edge)
    - Surface
- Collections
    - Polygon (area)

Common Functions:

1. Apply to all geometry types:
    a. SpatialReference
    b. Envelope
    c. Export
    d. IsSimple
    e. Boundary
2. Topological relationships:
    a. Equal
    b. Disjoint
    c. Intersect
    d. Touch - returns true if geometries touch but do not overlap
    e. Cross
    f. Within
    g. Contains
3. Spatial Data Analysis:
    a. Distance
    b. Buffer
    c. Union
    d. Intersection
    e. ConvexHull
    f. SymDiff