Relational Database Design - Lecture 5

Steps: ER → Schema

1. Regular Entity Types Mapping

~~2. Weak Entity Types~~

- Create ~~table~~ separate table for each regular entity types.
- Simple Attribute ⟶ Column
- Simple Components of composite ⟶ Column
- Multi-Valued ⟶ Refer to step 6
- Derived ⟶ Dont Add
- PK ⟶ (1 candidate key) will be selected and underlined
  or for composite attributes, underline all simple components

2. Weak Entity Type Mapping

- Follow all from step 1 except for PK.
- Owner's PK ⟶ Weak Entity's FK (shown as arrows)
  If this was composite, then add all as FK.
- → PK ⟶ FK + Weak Entity's Partial key
  - Underline both

3. Binary N:M Relationship Types Mapping

- Create a separate table for each N:M binary relationship
- PK of Connected Entities ⟶ FK in relationship table
- PK ⟶ The combined FK, and should be underlined
- Add Simple and Composite field of relationship like step 1.

Cross Referencing: Method of creating separate table for relationships

4. Binary 1 : N Relationship Mapping

- Cross Referencing is inefficient, use foreign key approach.
- For every 1 : N relationship
  PK of table in 1's side ⟶ FK of table in N's side
- Add relationship attributes to N side of relationship

5. 1 : 1 (binary) Relationship Mapping

- 3 Possible Approach:
  - Cross Referencing: Same step as step 3
  - Merged Relation Option:
    - Merge two entity tables into one
    - PK: Combination of two PK
    - May lead to data redundancy
  - Foreign key Approach: (Best Approach)
    - take any of the table's PK and add as FK for other table
      - Add as FK to the table which has a "total" participation, to prevent null values

Recursive Relationship

- Use same approach as binary 1 : N (step 4), 1 : 1 (step 5), N : M. (step 3)

6. Multivalued Attribute Mapping

- Each Multivalued Attribute (including multivalued composite) ⟶ Separate table
- PK of entity table ⟶ FK of new table
- Multivalued Only:
  - Add another column for storing value
  - Combined Primary key: FK and the column in prev line.

- Multivalued Composite:
  - Add only simple component of attribute as separate column
  - Combined PK: FK + Partial key (if exists)
  - If no PK: Underline all components as part of PK

7. N-ary Relationship Mapping

- Use cross-referencing method like step 3.

8. Mapping Specialization or Generalization

~~- Convert each specialization~~

Consider the subclass S1, S2, S3 and superclass C.

(a) Multiple relations - superclass and subclass: Works for all type
  C - Has the PK and all common fields
  S1, S2, S3 - Has the $PK_{(FK)}$ and the uncommon fields

(b) Multiple relations - subclass relations only: Only work for total
  C - No table
  S1, S2, S3 - Separate table with common field and uncommon fields

(c) Single relation with one type attribute: Disjoint
  C - Has all the fields (some will be null) of S1, S2, S3 and one ~~fie~~ attribute called a type [to denote ~~the~~ S1/S2/S3]

(d) Single relation with multiple type attributes: Overlap
  C - Will have all fields of S1, S2, S3. There will be S1Flag, S2Flag, S3Flag to denote overlap.

(e) Mapping of shared subclass (Multiple inheritance)
  You can use any of the options above (Note: The superclass may have the same PK)

Normalization - Lecture 6

A process used to make database more effecient.

Informal Guideline - No written rules.

~~Formal Guideline~~

Normal Forms

- 1st NF, 2nd NF, ~~3rd NF,~~ BNCF NF, 3rd NF, 4th NF, 5th NF

1st Normal Form

- Disallows
  - Composite Attribute
  - Multivalued Attribute
  - Nested Relation; Attributes whose value for an individual tuple are non-atomic
- Considered to be a part of the definition of relation.

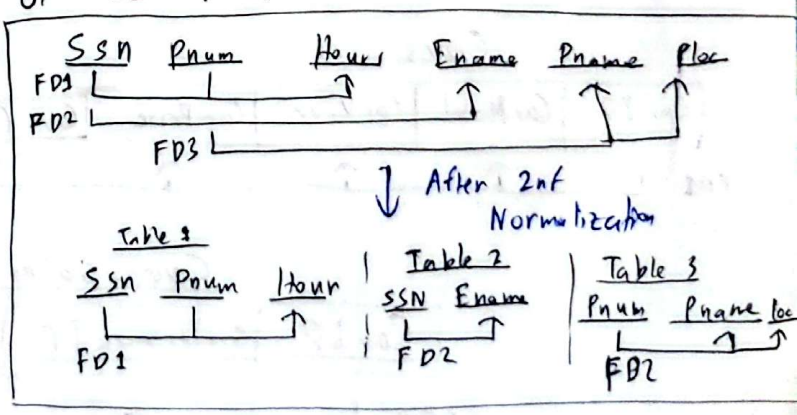
2nd Normal Form

- **Prime Attribute**: An attribute that is member of ~~the primary~~ candidate key K.
- **Full Functional Dependency**: A FD (functional dependency) $Y \rightarrow Z$ where removal of any atribute from Y means the FD does not hold any more.
  - One or more attributes depend fully on the PK

Partial Functional Dependency: One or more attribute depend ~~partly~~ on ~~the PK~~ a part of the PK.

Candidate key: Every key with potential to be PK

Secondary key: Candidate key not chosen as PK



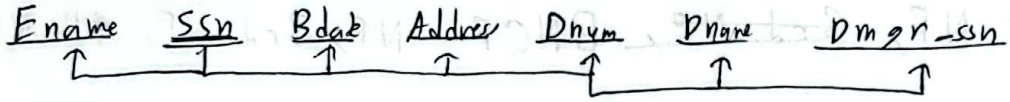
3rd Normal Form

Transitive FD: If $X \to Y$, $Y \to Z$ then $X \to Z$

Conditions

- It must be in 2 NF
- There ~~must~~ cannot be any transitive FD.

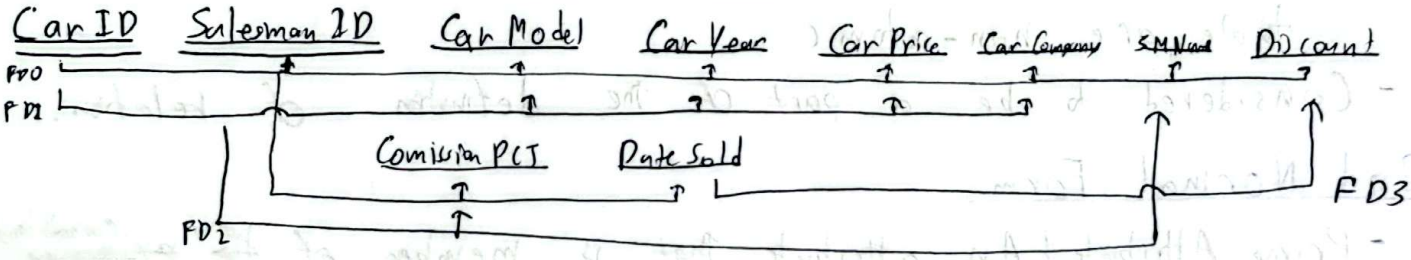
Example



↓ After 3NF

Question Eg.

Car Sales Table



Q) Is this in 1st NF

- Yes, There are no multivalued, composite attribute or nested relation.

Some ~~Do~~ Notation

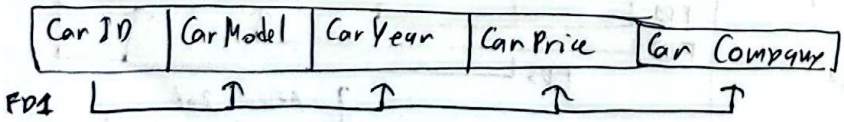
Car Model(Name, Year) Composite

{Car Model} - Multivalued

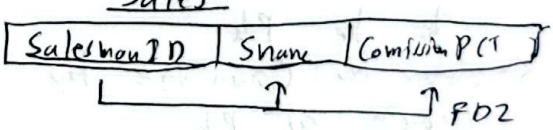
Q) Is this in 2nd NF? Why or why not. Convert to 3rd NF.

It is not in 2 NF as, FD1, FD2 has partial ~~key~~ ~~dependency~~ functional dependency.

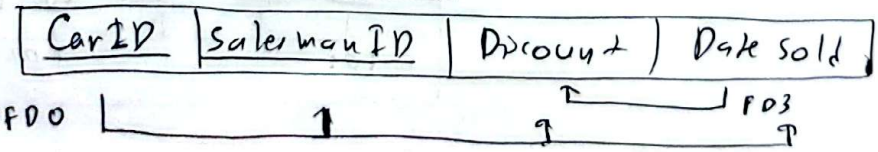
Cars



Sales

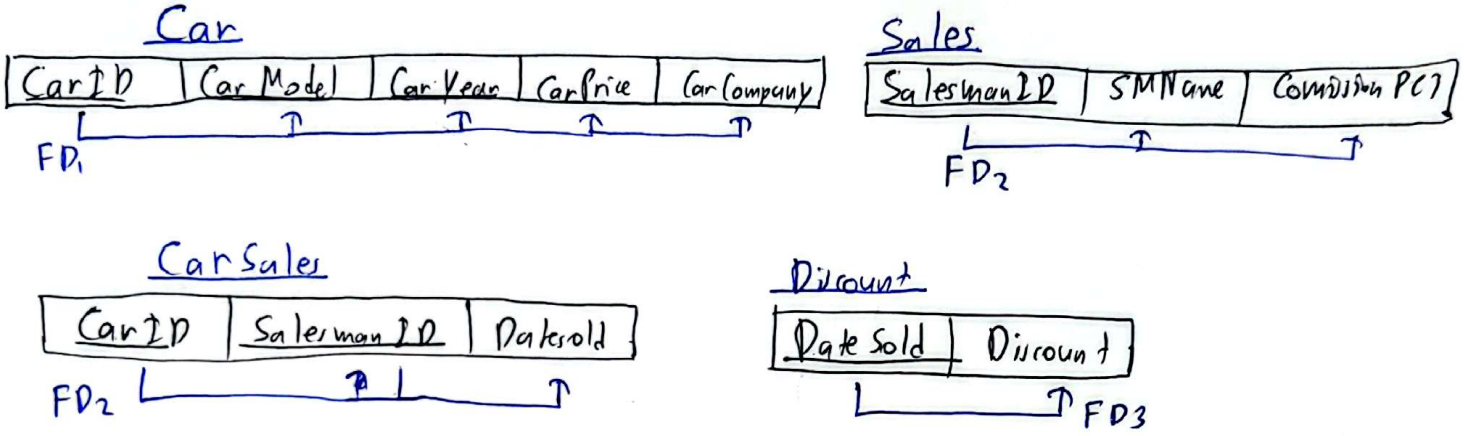


Car_Sales



Q) Explain if the relational schema is in 3NF or not. If not, normalize to 3NF.

- It is not in 3NF as there is transitive dependency, FD3.



Note: BCNF is not in syllabus

Superkey: Set of attribute which will be unique accross all rows

- Any ~~sup~~ key K is a superkey if removing any attribute from K will cause K not to be a superkey any more.

Indexing in Database - Lecture 7

- An Index in database works in same way as textbook
- Speeds up access to desired data

Search key: Attribute or set of attribute used to look up records in a file.

Index File: | search-key | pointer |

- Index files are smaller than original file

Types of indices:

- Ordered indices: Clustering Index (primary) / Nonclustering index (secondary)
- Hash indices: Search keys are distributed uniformly across "buckets" using a "hash function"

Ordered Indices

- Index entries are stored sorted on the search key value

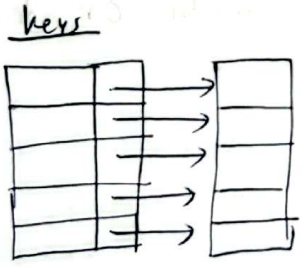
Clustering Index: In a sequentially ordered file, the index whose search key specifies the sequential order of the file

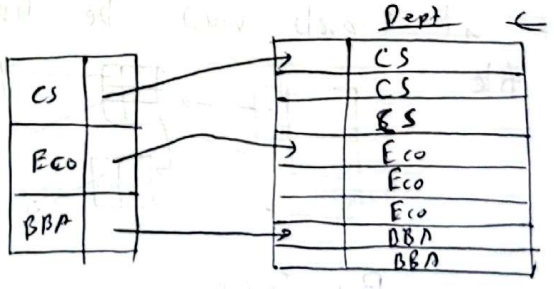
Secondary Index: An index whose search key specifies an order different from the sequential order of the file.

Index-sequential file: Sequential file ordered on a search key, with a clustering index on the search key.

Dense Index Files

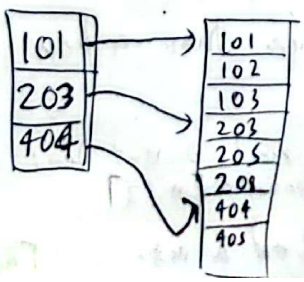
Index record appears for every search-key value in the file





Sparse Index Files

- Contain index records for only some search-key
- Applicable when records are stored sequentially



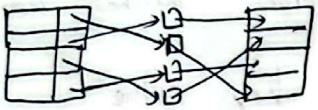
- Searching ~~205~~ k:
  - Find largest key < k
  - Sequentially continue

- Less space and less maintenance overhead for insertions and deletions
- Generally slower than dense index for locating records.
- Good Tradeoff:

For Clustered Index: Sparse index with an index entry for every block in file corresponding to least search-key value in the block

For Unclustered Index: Sparse index on top of dense index (multilevel index)

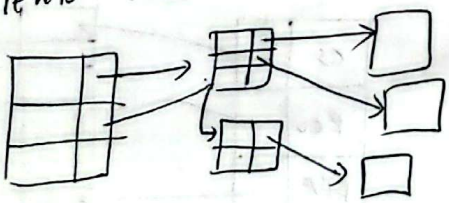
Secondary Indices



- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value
- Secondary indices have to be dense

Multilevel Index

- If index does not fit in memory, access becomes expensive
- Solution: Treat index kept on disk as a sequential file and construct a sparse index on it.
  - Outer Index - A sparse index of the basic index
  - Inner Index - The basic index file
- Even if outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.



B+ Tree

- Every tree has an order of "n"
- All node on the left have smaller values than a key and all nodes on the right have greater than or equal values than a key.

Each node has	Max	Min
• Children • Keys/values	• root/internal: n leaf: 0 • n-1	• Root: 2 (if root is not leaf as well) Internal: $\lceil n/2 \rceil$ • Root: 1, Leaf & internal: $\lceil n/2 \rceil - 1$

Note: For simplicity leaf and internal is considered same in some resources, otherwise leaf min is $\lceil (n-1)/2 \rceil$

B+ Tree (Insertion)

1) Find the right spot (leaf node): Start from root and move down the tree, following the correct path (based on the key you want to insert), until you reach a leaf.
2) If node has less than "max" keys:
   - Insert value in sorted order
3) If node has "max" keys:
   - Split into half
   - 1st half: new left leaf
   - 2nd half: New right leaf
   - Parent: lowest key from right node
4) Repeat split upwards if needed.
   - If parent now becomes full too, split it as well and push a key further up.
   - If a non-leaf node is split, the lowest value on the right will only be moved up and not copied to the new right leaf. This might continue up to the root.
   - If root splits → new root will be created and the tree height will increase.

- If order of B+ tree is odd, then create a right-biased or left-biased tree. Maintain the same bias for entire tree.

B+ Tree (Deletion)

- Use common sense and make sure the properties of B+ tree are maintained

B+ Property

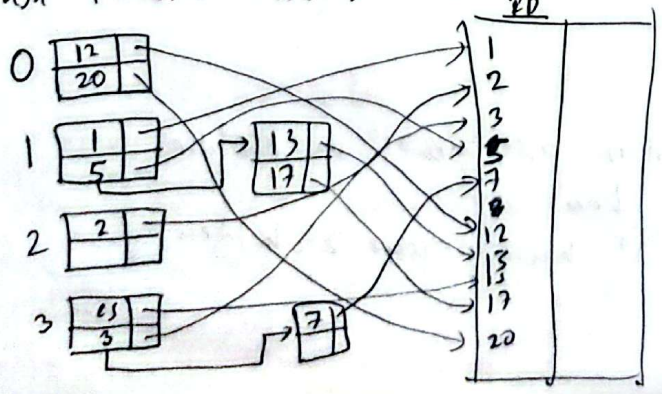
- Balanced: All leaf at same level
- Order m: Each internal node has at most m children
- Leaf stores at most m-1 keys
- If root is not leaf, it has atleast 2 children

Static Hashing

- **Bucket**: Unit of storage containing one or more entries (a bucket)
  - Obtained from an entry's search-key value using a hash function
- **Hash Function, h**: Set of all search-key values $K \rightarrow$ Set of all bucket addresses B.
  - Used to locate entries for access, insertion as well as deletion
- Entries with different search-key values may be mapped to the same bucket, thus entire bucket has to be searched sequentially to locate an entry.
- hash index: buckets store entries with pointers to records
- hash file organization: buckets store records
- Bucket overflow can occur because of
  - Insufficient buckets
  - Skew in distribution records. This can occur due to two reasons:
    - Multiple records have same search-key value
    - Chosen hash function produces non-uniform distribution of key values
- Although the probability of bucket overflow can be reduced, it cannot be eliminated; it is handled by using overflow buckets
- **Overflow Chaining** - The overflow buckets of a given bucket are chained together in a linked list
- Above scheme is called closed addressing
- An alternative, called open addressing which does not use overflow buckets, is not suitable for database applications.

Example

Hash Function $h(ID) = (\text{sum of digits in } ID) \% 4$



Transaction, Concurrency Control & Recovery

Transaction: Logical unit of database processing that includes one or more access operations

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 50 WHERE account-id = 'A';
                              + 50          "             = 'B';
COMMIT
```

- Transaction may be stand-alone or may be embedded within a program

ACID

- Used to preserve the integrity of data enforced by Concurrency Control and recovery methods.
- **Atomicity**: A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all. Ensured by ~~recovery~~ transaction recovery subsystem.
- **Consistency**: A transaction should be consistency preserving, meaning if it is completely executed from beginning to end, it should take the database from one consistent state to another. maintained by both recovery and concurrency.
- **Isolation**: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. A transaction should not make its updates visible to other transactions until it is committed
  - Ensured by running transaction serially (concurrency control)
- **Durability**: The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure
  - Ensure by recovery subsystem of DBMS

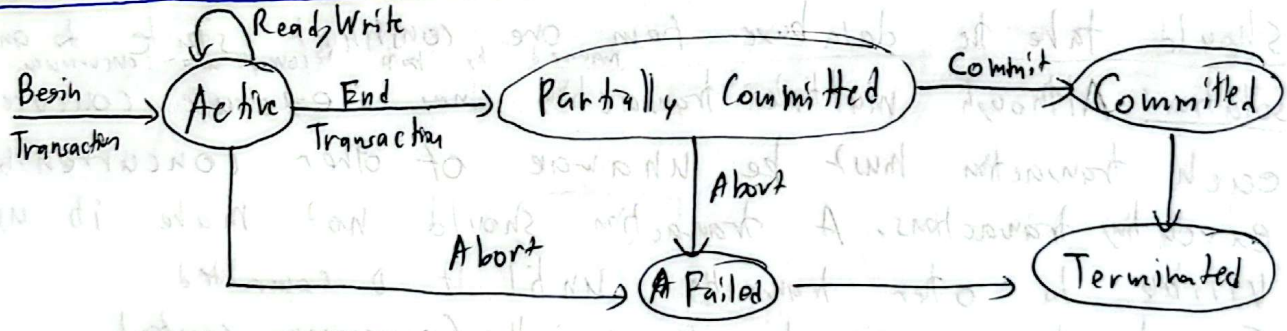
Issues that can occur for Transaction

- Recovery: Take DB to previous stable state
- Concurrency Control: Concurrent execution of multiple transactions

Why Recovery is Needed

- Computer Failure (system crash)
- A transaction or system error: Logical error, erroneous param, interrupt
- Local errors or exceptions detected by the transaction
  - Certain conditions necessitate cancellation of transaction
    - Data of transaction not found; insufficient balance
    - A programed abort in the transaction.
- Concurrency control enforcement:
  Concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in the state of deadlock
- Disk Failure
- Physical Problem and Catastrophes: Includes power, fire, etc

Recovery (Transaction States)



Concurrency Control

- Multiple transaction are allowed to run concurrently in the system
- Advantages:
  - Increased processor and disk utilization
  - Reduced average response time for transactions

Concurrency Control Schemes: Mechanisms to achieve isolation to prevent transaction from destroying consistency of database.

Issues that might be caused in absence of Concurrency Control

- **Lost Update Problem:** Two transaction updating same data, causing one getting overwritten.
- **Dirty Read (Temporary Update) problem:** Transaction reading ~~uncommitted~~ data written by uncommitted transaction and the ~~transac~~ uncommitted transaction gets reverted.
- **The Incorrect Summary (or Incorrect Analysis) Problem:** A transaction performs an aggregate function while other transactions are updating data. The aggregate function may calculate some values before they are updated and others after they are updated.
- **Non Repeatable Read:** A transaction reads the same row twice and get different value to to another transaction.
- **Phantom Read:** A transaction re executing a query ~~bec getting~~ returning a set of rows, rows appear or disappear due to another transaction's insert or deletes.

Transaction in SQL

- SQL statements are always atomic
- End Statements: COMMIT, ROLLBACK
  - By default. Can be turned off.
- Isolation level can be changed at the start of transaction, or be set at database level.

Isolation Levels

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
Serializable: Transactions happen in a serial order.	Prevents	Prevents	Prevents
Repeatable Read: Only comitted records can be read. Repeated read, returns same value	Prevents	Prevents	Allows
Read Committed: Only committed records can be read.	Prevents	Allows	Allows
Read Uncommitted: Even uncommitted records can be read.	Allows	Allows	Allows