Data Engineering Techniques (AID30010) — Unit III

Mining Intelligent Data Patterns

Structured Question Bank

PART B: LONG ANSWER QUESTIONS (5 marks)

Association Rules & Market Basket Analysis

Q1: What is Market Basket Analysis? Explain its real-life applications.

Answer:

1. Definition — Market Basket Analysis

- Market Basket Analysis (MBA) is a data mining technique used to discover relationships between items frequently purchased together
- It analyzes customer purchase patterns to find which products co-occur in the same transaction
- Based on the concept of **association rule mining**
- The "basket" refers to all items a customer places in their shopping cart in a single transaction
- The goal is to find rules like: *"Customers who buy X also tend to buy Y"*

2. Key Terms

- **Itemset:** A collection of one or more items — e.g., {Bread, Milk}
- **Transaction:** A single purchase event containing a set of items
- **Frequent Itemset:** An itemset that appears in at least a minimum number of transactions (minimum support threshold)
- **Association Rule:** An implication of the form X → Y (if X is bought, Y is also likely bought)
- **Rule Generation:** After finding frequent itemsets, rules with sufficient confidence are extracted

3. Real-Life Applications

- **Retail / Supermarkets:** Place related products near each other — e.g., chips near beverages; Amazon's "Frequently Bought Together" feature
- **E-Commerce / Recommendation Systems:** Netflix, Amazon, Flipkart use MBA to recommend products/movies based on purchase/watch history

- **Healthcare:** Identify combinations of symptoms, diagnoses, or treatments that co-occur frequently
- **Banking / Finance:** Detect patterns in financial transactions — e.g., customers who apply for home loans also inquire about insurance
- **Telecommunications:** Identify which service bundles customers subscribe to together for cross-selling

4. Why Market Basket Analysis is Important

- Increases sales through product bundling and cross-selling strategies
- Helps in store layout optimization — high co-occurring items placed strategically
- Supports targeted marketing and personalized promotions
- Reduces inventory waste by understanding demand patterns
- Powers recommendation engines that improve customer experience

5. How it Works — Overview

- Collect transactional data (TID + items in each transaction)
- Define minimum support and minimum confidence thresholds
- Apply an algorithm (Apriori or FP-Growth) to find frequent itemsets
- Generate association rules from frequent itemsets
- Evaluate rules using support, confidence, and lift

Q2: Explain Support, Confidence, and Lift with formulas and examples.

Answer:

1. Why These Measures?

- Association rules are evaluated using three key interestingness measures
- A rule must pass minimum thresholds of support and confidence to be considered "strong"
- Lift is used to confirm whether the rule is genuinely meaningful or just a coincidence
- Together, these three measures help filter out weak/misleading rules
- High confidence alone is NOT enough — lift must also be checked

2. Support

- Measures how frequently an itemset appears in the entire dataset
- Tells us: *"How common is this combination?"*

Formula	Meaning
Support(A) = Count(A) / Total Transactions	Frequency of A alone
Support(A ∪ B) = Count(A and B together) / Total Transactions	Frequency of A and B together

- Example: If {Bread, Milk} appears in 3 out of 5 transactions → Support = 3/5 = **60%**
- Low support → itemset is rare → usually not useful
- Used to filter out infrequent itemsets early in the process

3. Confidence

- Measures the reliability/strength of the rule X → Y
- Tells us: *"Given that X was bought, how often is Y also bought?"*

Formula
Confidence(A → B) = Support(A ∪ B) / Support(A)

- Example: Support({Bread, Milk}) = 60%, Support({Bread}) = 80% → Confidence(Bread → Milk) = 60% / 80% = **75%**
- High confidence = rule is reliable
- Limitation: Can be misleading if item B is very common in the dataset

4. Lift

- Measures the true strength of association between A and B — beyond random chance
- Tells us: *"How much more likely is B bought when A is bought, compared to if A wasn't considered?"*

Formula
Lift(A → B) = Confidence(A → B) / Support(B)
OR equivalently: Lift(A → B) = Support(A ∪ B) / (Support(A) × Support(B))

Interpretation:

Lift Value	Meaning
Lift > 1	Positive association — A and B occur together more than expected ✅
Lift = 1	Independent — no real association
Lift < 1	Negative association — A and B avoid each other ❌

5. Key Insight (Very Important for Exams)

- High Confidence ≠ Strong Rule
- Always check: Support → Confidence → **Lift** (most important for true insight)
- Example: Bread → Milk (Confidence = 75%, Lift = 0.94) → **Weak rule** despite good confidence
- Example: Butter → Milk (Confidence = 100%, Lift = 1.25) → **Strong rule** ✅
- Lift corrects the bias that comes from very common items inflating confidence

---

Q3: Explain Frequent Itemsets and Closed Itemsets.

Answer:

1. Frequent Itemset — Definition

- An itemset is called **frequent** if its support is greater than or equal to the minimum support threshold (min_sup)
- Notation: L1 = frequent 1-itemsets, L2 = frequent 2-itemsets, and so on
- Example: If min_sup = 50% and {Bread, Milk} appears in 3/5 transactions (60%), then {Bread, Milk} is a frequent itemset
- Goal: Find all itemsets that satisfy the minimum support threshold
- Frequent itemsets are the foundation for generating association rules

2. Apriori Property (Anti-Monotone Property)

- *"If an itemset is frequent, then ALL of its subsets must also be frequent"*
- Conversely: *"If an itemset is infrequent, then ALL of its supersets are also infrequent"*
- This is called the **downward closure property**
- This property allows Apriori algorithm to prune (cut) large portions of the search space
- Example: If {A, B} is infrequent → {A, B, C}, {A, B, D}, etc. are all infrequent and can be skipped

3. Closed Itemset — Definition

- An itemset X is **closed** if there is NO superset of X that has the same support as X
- In other words, X is closed if adding any more items to X would reduce the support count
- Closed itemsets represent a compact, lossless representation of all frequent itemsets
- Every frequent itemset can be derived from a closed frequent itemset
- Example: If Support({A, B}) = Support({A, B, C}), then {A, B} is NOT closed; {A, B, C} is closed

4. Maximal Itemset — Definition

- An itemset X is **maximal frequent** if it is frequent AND no superset of X is also frequent
- Maximal itemsets are a subset of closed itemsets
- They provide the most compact summary but are **lossy** — you cannot recover exact support of subsets
- Difference: Closed → lossless compression; Maximal → lossy compression
- Used when only a boundary representation is needed

5. Comparison Table

Property	Frequent Itemset	Closed Itemset	Maximal Itemset
Support condition	⩾ min_sup	⩾ min_sup	⩾ min_sup
Superset condition	None	No superset with same support	No frequent superset
Information loss	No	No (lossless)	Yes (lossy)
Compactness	Least compact	Compact	Most compact
Use case	Foundation for rules	Efficient mining	Boundary representation

Apriori Algorithm

Q4: Explain the Apriori Algorithm in detail with a complete example.

Answer:

1. Definition

- Apriori is a classical algorithm for mining frequent itemsets and generating association rules from transactional datasets

- Proposed by **Agrawal and Srikant** in 1994
- Uses the **Apriori Property** (anti-monotone property) to prune the search space
- Works in a **level-wise (breadth-first)** manner — finds all frequent k-itemsets before moving to (k+1)-itemsets
- Widely used in Market Basket Analysis, recommendation systems, and sales pattern analysis

2. Key Terms

- **min_sup:** Minimum support threshold — itemsets below this are discarded
- **min_conf:** Minimum confidence threshold — rules below this are discarded
- **Candidate Generation (Ck):** All candidate k-itemsets generated from frequent (k-1)-itemsets
- **Pruning:** Removing candidates whose subsets are not frequent
- **Lk:** Set of frequent k-itemsets after scanning the database

3. Algorithm — Step-by-Step

1. Scan the database → count support of each item → generate L1 (frequent 1-itemsets)
2. Use Lk−1 to generate Ck (candidates for k-itemsets) — join step
3. **Pruning:** Remove candidates from Ck if any (k-1)-subset is not in Lk−1
4. Scan database → count support for remaining Ck → generate Lk (items that meet min_sup)
5. Repeat until no new frequent itemsets are found
6. Generate association rules from all frequent itemsets using min_conf threshold

4. Complete Example

- Transaction Data, min_sup = 50%, min_conf = 75%, Total Transactions = 5

TID	Items
T1	Bread, Milk
T2	Bread, Diaper, Beer, Eggs
T3	Milk, Diaper, Beer, Coke
T4	Bread, Milk, Diaper, Beer
T5	Bread, Milk, Diaper, Coke

Step 1: L1 — Frequent 1-itemsets

Item	Count	Support	Frequent?
Bread	4	80%	✅
Milk	4	80%	✅
Diaper	4	80%	✅
Beer	3	60%	✅
Coke	2	40%	❌
Eggs	1	20%	❌

L1 = {Bread}, {Milk}, {Diaper}, {Beer}

Step 2: L2 — Frequent 2-itemsets (from C2)

Itemset	Count	Support	Frequent?
{Bread, Milk}	3	60%	✅
{Bread, Diaper}	3	60%	✅
{Bread, Beer}	2	40%	❌
{Milk, Diaper}	3	60%	✅
{Milk, Beer}	2	40%	❌
{Diaper, Beer}	3	60%	✅

L2 = {Bread, Milk}, {Bread, Diaper}, {Milk, Diaper}, {Diaper, Beer}

Step 3: L3 — Frequent 3-itemsets (from C3)

Itemset	Count	Support	Frequent?
{Bread, Milk, Diaper}	2	40%	❌

No frequent 3-itemsets. Algorithm stops.

Step 4: Generate Association Rules (from L2)

From {Bread, Milk} (Support = 60%):

- Bread → Milk: Conf = 60/80 = 75% ✅
- Milk → Bread: Conf = 60/80 = 75% ✅

From {Diaper, Beer} (Support = 60%):

- Diaper → Beer: Conf = 60/80 = 75% ✅
- Beer → Diaper: Conf = 60/60 = 100% ✅

5. Advantages and Disadvantages of Apriori

Advantages	Disadvantages
Simple and easy to understand	Requires multiple database scans (one per level)
Uses Apriori property for pruning	Generates a large number of candidate itemsets
Works well for sparse datasets	Slow for large datasets or low min_sup
Easily generates association rules	High memory usage for storing candidates
Well-studied and widely implemented	Performance degrades with many items or transactions

---

Q5: Explain how to Generate Association Rules from Frequent Itemsets.

Answer:

1. What is Rule Generation?

- After all frequent itemsets are found, the next step is to extract association rules from them
- For each frequent itemset L, generate all non-empty subsets of L
- For each non-empty subset S of L, output the rule: S → (L − S) if Confidence ⩾ min_conf
- The confidence of the rule S → (L − S) = Support(L) / Support(S)
- Since L is frequent, Support(L) is already known — only confidence needs to be checked

2. Rule Generation — Steps

1. Take each frequent itemset L (length ⩾ 2)
2. Enumerate all non-empty subsets of L
3. For each subset S, form the rule: S → (L − S)
4. Calculate: Confidence = Support(L) / Support(S)
5. If Confidence ⩾ min_conf → output the rule as a **strong association rule**

3. Example

- Frequent itemset: {Bread, Milk, Diaper} with Support = 40%
- Possible rules (non-empty subsets as antecedents):

Rule	Confidence Calculation
Bread → Milk, Diaper	Support({B,M,D}) / Support({B}) = 40/80 = 50%
Milk → Bread, Diaper	40/80 = 50%
Diaper → Bread, Milk	40/80 = 50%
Bread, Milk → Diaper	40/60 = 67%
Bread, Diaper → Milk	40/60 = 67%
Milk, Diaper → Bread	40/60 = 67%

- Rules below min_conf (e.g., 75%) are discarded

4. Anti-Monotone Property for Confidence

- If a rule X → Y−X does NOT satisfy min_conf, then any rule X' → Y−X' where X' ⊂ X also won't satisfy it
- This allows pruning of rules just like itemsets are pruned in the Apriori algorithm
- Reduces the number of rules to evaluate significantly

5. Important Exam Points

- Every association rule comes from a frequent itemset
- The antecedent and consequent together form a frequent itemset
- Support of a rule = Support of its corresponding itemset
- Confidence uses support of the antecedent (already known from frequent itemset mining)
- A strong rule must satisfy BOTH min_sup (via its itemset) AND min_conf

---

Q6: Explain the methods to Improve the Efficiency of Apriori.

Answer:

1. Why Efficiency Improvement is Needed

- Apriori can be slow for large datasets because it scans the database multiple times
- It generates a large number of candidate itemsets which consume memory and time
- For low minimum support values, the number of frequent itemsets can explode

- Multiple I/O scans are expensive for very large transaction databases
- Several techniques have been developed to overcome these bottlenecks

2. Hash-Based Technique

- Use a hash table to quickly count candidate 2-itemsets during the L1 scan
- If the hash bucket count for a pair is below min_sup, that pair is NOT frequent → remove from C2
- Reduces the size of C2 significantly before the actual support counting pass
- Only one extra pass is needed — no additional database scans
- Effective for reducing large candidate sets early in the process

3. Transaction Reduction

- A transaction that does NOT contain any frequent k-itemset cannot contribute to frequent (k+1)-itemsets
- Such transactions can be **marked and removed** from subsequent database scans
- Reduces the effective size of the database with each iteration
- Particularly useful when many transactions become irrelevant in higher iterations
- Improves speed significantly for sparse datasets

4. Partitioning

- Divide the database into non-overlapping partitions that fit in memory
- Mine each partition independently to find **locally frequent** itemsets
- Any globally frequent itemset must be locally frequent in at least one partition
- Merge all local frequent itemsets → candidate global frequent itemsets
- Requires only **2 database scans** regardless of the number of levels — very efficient

5. Sampling

- Take a random sample of the database and mine frequent itemsets from it
- Use a **lower min_sup threshold** on the sample to avoid missing globally frequent itemsets
- Verify results with one more pass on the entire database
- May miss some globally frequent patterns (false negatives) but extremely fast
- Good for exploratory data mining where approximation is acceptable

6. Dynamic Itemset Counting (DIC)

- Divide the database into blocks (marked with start points)
- Add new candidate itemsets at any start point — not just at the beginning of each pass
- A candidate can be added when its subsets are all found to be frequent
- Reduces the number of database passes needed

- More flexible than standard Apriori — avoids waiting for a full scan to complete

---

FP-Growth Algorithm

Q7: Explain the FP-Growth Algorithm in detail — Definition, Working, and Example.

Answer:

1. Definition

- FP-Growth (Frequent Pattern Growth) is an efficient algorithm for mining frequent itemsets **without generating candidate itemsets**
- Proposed by **Han, Pei, and Yin** in 2000
- Uses a compact tree structure called an **FP-Tree (Frequent Pattern Tree)** to store the database
- Requires only **2 database scans** — regardless of the number of frequent itemsets
- Much faster and more memory-efficient than Apriori for large datasets

2. Key Idea

- Instead of generating and testing candidate itemsets (like Apriori), FP-Growth directly mines patterns from a compressed FP-Tree
- The FP-Tree captures all transaction information in a compact structure
- Patterns are mined by building **Conditional FP-Trees** recursively
- Avoids the expensive candidate generation step completely
- The Apriori property is still used implicitly — but no explicit candidate lists are created

3. Algorithm — Step-by-Step

Step 1: Scan database → count item frequencies

- Remove items below min_sup (infrequent items)
- Create a **frequency-ordered list** (most frequent to least frequent)

Step 2: Reorder transactions

- Sort items within each transaction according to the global frequency order (descending)
- Remove infrequent items from transactions

Step 3: Build FP-Tree

- Start with a NULL root node
- Insert each reordered transaction one by one into the tree

- Shared prefixes are merged → common paths share nodes
- Each node stores: item name + count
- A **Header Table** stores pointers to all nodes of each item (for traversal)

Step 4: Mine FP-Tree (Pattern Extraction)

- Start from the **least frequent item** in the header table
- Find all paths from root to that item → these are the **prefix paths / conditional pattern base**
- Build a **Conditional FP-Tree** from the conditional pattern base
- Mine the conditional FP-Tree recursively to generate all frequent patterns involving that item
- Repeat for every item in the header table (bottom to top)

4. Complete Example

- Transaction data, Minimum Support = 3

TID	Items	Reordered (Q>S>P>T>R)
T1	S, Q	Q, S
T2	R, Q, T, P, S	Q, S, P, T, R
T3	S	S
T4	P, S, T	S, P, T
T5	R, P, Q, T	Q, P, T, R
T6	T, P, S, Q	Q, S, P, T
T7	Q, P, S	Q, S, P
T8	Q, R	Q, R

Item Frequency Count:

Item	Frequency	Frequent?
Q	6	✅
S	6	✅
P	5	✅
T	4	✅
R	3	✅
U	1	❌

Header Table: Q(6) → S(6) → P(5) → T(4) → R(3)

FP-Tree Structure (After inserting all transactions):

```
NULL
├── Q(6)
│   ├── S(4)
│   │   ├── P(3)
│   │   │   └── T(2)
│   │   │       └── R(1)
│   │   └── (leaf)
│   ├── P(1)
│   │   └── T(1)
│   │       └── R(1)
│   └── R(1)
└── S(2)
    └── P(1)
        └── T(1)
```

Step 5: Mine Patterns for R (least frequent)

- Conditional Pattern Base for R: {Q,S,P,T:1}, {Q,P,T:1}, {Q:1}
- Frequent patterns found: {R}, {Q,R}, {P,R}, {T,R}, {Q,P,R}, {Q,T,R}, {P,T,R}, etc.

5. Advantages and Disadvantages of FP-Growth

Advantages	Disadvantages
No candidate generation	FP-Tree can be large for dense datasets
Only 2 database scans needed	Building FP-Tree requires memory
Much faster than Apriori	More complex to implement
Memory-efficient with tree compression	Conditional FP-Trees add overhead
Works well for large, sparse datasets	Less intuitive than Apriori

Q8: Compare Apriori and FP-Growth Algorithm.

Answer:

1. Core Difference

- Apriori generates candidate itemsets explicitly and tests them against the database
- FP-Growth avoids candidate generation by compressing the database into an FP-Tree
- Both find the same frequent itemsets but take fundamentally different approaches
- Apriori is simpler to understand; FP-Growth is significantly faster in practice
- The choice depends on dataset size, density, and available memory

2. Comparison Table

Feature	Apriori	FP-Growth
Candidate Generation	Yes (explicit Ck generation)	No
Database Scans	Multiple (one per level k)	Only 2
Data Structure	Itemset lists	FP-Tree
Approach	Level-wise (breadth-first)	Divide and conquer (recursive)
Speed	Slow for large datasets	Fast
Memory Usage	High (stores all candidates)	Efficient (compressed tree)
Scalability	Poor for low min_sup	Good
Complexity of Implementation	Simple	More complex
Pruning Method	Anti-monotone property	Conditional FP-Tree
Best For	Small datasets, high min_sup	Large datasets, low min_sup

3. When to Use Which

Scenario	Recommended Algorithm
Small dataset	Apriori (simple to understand)
Large dataset	FP-Growth
Low minimum support threshold	FP-Growth
Need to explain steps to non-technical audience	Apriori
Memory-constrained environment	Apriori (no tree structure needed)
Dense dataset with many frequent patterns	FP-Growth

4. Similarity

- Both use the Apriori property (anti-monotone downward closure)
- Both require min_sup and min_conf thresholds
- Both produce the same final set of frequent itemsets
- Both are followed by the same association rule generation step
- Both are used for Market Basket Analysis and similar applications

5. Key Exam Point

- FP-Growth is **always preferred in practice** for large-scale data
- Apriori is important to understand because it introduces the fundamental concepts
- FP-Growth's advantage: *"No candidate generation + only 2 scans"* — remember this line for exams

---

Additional Practice Questions

Q9: Define Association Rule. Explain the types of association rules and properties.

Answer:

1. What is an Association Rule?

- An association rule is an implication of the form: **X → Y**
  - X = antecedent (left-hand side) — the condition
  - Y = consequent (right-hand side) — the result
- Both X and Y are itemsets; they are disjoint: $X \cap Y = \varnothing$
- Meaning: *"If a customer buys items in X, they are also likely to buy items in Y"*
- Example: {Bread, Butter} → {Jam} means customers who buy Bread and Butter also tend to buy Jam
- Evaluated using Support, Confidence, and Lift

2. Types of Association Rules

- **Single-Dimensional Rules:** Involve only one attribute/dimension
  - Example: buys(X, "bread") → buys(X, "milk")
- **Multi-Dimensional Rules:** Involve two or more dimensions/attributes
  - Example: age(X, "30–40") ∧ income(X, "high") → buys(X, "laptop")
- **Multi-Level Rules:** Rules mined at different levels of a concept hierarchy
  - Example: buys(X, "2% milk") → buys(X, "wheat bread") [specific level]
  - Example: buys(X, "milk") → buys(X, "bread") [higher level]
- **Quantitative Rules:** Involve quantitative (numeric) attributes
  - Example: age(X, "30–40") → buys(X, "smartphone")

3. Properties of Association Rules

- **Support:** Must be ⩾ min_sup for the itemset to be considered
- **Confidence:** Must be ⩾ min_conf for the rule to be considered strong
- **Lift:** Must be > 1 for the rule to represent a genuine positive association
- **Downward Closure (Apriori Property):** Support of any itemset ⩽ Support of its subsets

- **Rule Monotone Property for Confidence:** If X → Y−X fails min_conf, subsets of X as antecedents will also fail

4. Interesting Rules vs. Trivial Rules

- A rule is **interesting** if it is: unexpected, actionable, or confirms a valuable hypothesis
- A rule is **trivial** if it simply confirms the obvious (e.g., obvious product combinations)
- Lift helps measure interestingness — Lift > 1 means the rule is more interesting than random
- **Null-invariance** is important — measures should not be affected by transactions that contain neither X nor Y
- Cosine, Jaccard, and All-Confidence are null-invariant alternatives to standard measures

5. Steps from Data to Rules — Full Pipeline

Step	Action
1	Collect transaction data
2	Set min_sup and min_conf thresholds
3	Find all frequent itemsets (Apriori / FP-Growth)
4	Generate candidate rules from frequent itemsets
5	Filter rules by min_conf
6	Evaluate rules using Lift for interestingness
7	Present actionable strong rules to business users

Q10: Explain Mining Frequent Itemsets Without Candidate Generation (FP-Growth approach).

Answer:

1. The Problem with Candidate Generation

- Apriori generates a huge number of candidate itemsets in each iteration
- For k-itemsets, the number of candidates can be exponentially large
- Each candidate set requires an expensive database scan to count support
- The candidate generation + testing cycle is a major bottleneck for large datasets
- Han et al. proposed FP-Growth in 2000 to completely eliminate candidate generation

- Instead of generating candidates, **grow patterns directly from the database**
- Compress the entire transaction database into an FP-Tree (two database scans)
- Mine frequent patterns by traversing the FP-Tree — no database scans needed during mining
- Divide and conquer: break the problem into smaller sub-problems using conditional FP-Trees
- Each sub-problem is independent and can be solved recursively

3. Conditional Pattern Base

- For each frequent item X (processed from least frequent to most frequent):
  - Find all paths in the FP-Tree from root to X → these are the **prefix paths**
  - Each prefix path forms one entry in the **Conditional Pattern Base** for X
  - The support of each prefix path = the count of node X at the end of the path
- Example: For item R with count 3 in the tree, the conditional pattern base contains all paths that end at R

4. Conditional FP-Tree

- Build a new FP-Tree from the conditional pattern base of X
- This is the **Conditional FP-Tree** for X
- Mine this smaller tree recursively to find all frequent patterns containing X
- Patterns that emerge from the conditional FP-Tree + X = frequent patterns involving X
- Process continues recursively until the conditional FP-Tree is empty or a single path

5. Advantages Over Apriori

- No candidate itemset generation — eliminates the biggest bottleneck
- Only 2 full database scans (once for counting, once for tree construction)
- All subsequent mining is done in memory on the tree structure
- FP-Tree compresses the database — common prefixes are shared (not repeated)
- Highly scalable — performance gap over Apriori widens as dataset grows larger

---

Q11: Solve a complete Apriori Algorithm problem with Support and Confidence calculation.

Answer:

Problem: Apply Apriori Algorithm on the following data. Find frequent patterns step by step. min_sup = 2 (count-based, 40%), min_conf = 75%

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Total Transactions = 5

Step 1: L1 — Frequent 1-itemsets (min count = 2)

Item	Count	Frequent?
Bread	4	✅
Milk	4	✅
Diaper	4	✅
Beer	3	✅
Eggs	1	❌
Coke	2	✅

L1 = {Bread}, {Milk}, {Diaper}, {Beer}, {Coke}

Step 2: L2 — Frequent 2-itemsets

Itemset	Count	Frequent?
{Bread, Milk}	3	✅
{Bread, Diaper}	3	✅
{Bread, Beer}	2	✅
{Bread, Coke}	1	❌
{Milk, Diaper}	3	✅
{Milk, Beer}	2	✅
{Milk, Coke}	2	✅
{Diaper, Beer}	3	✅
{Diaper, Coke}	2	✅
{Beer, Coke}	1	❌

L2 = {B,M}, {B,D}, {B,Beer}, {M,D}, {M,Beer}, {M,Coke}, {D,Beer}, {D,Coke}

Step 3: L3 — Frequent 3-itemsets (candidates from L2)

Itemset	Count	Frequent?
{Bread, Milk, Diaper}	2	✅
{Bread, Milk, Beer}	2	✅
{Bread, Diaper, Beer}	2	✅
{Milk, Diaper, Beer}	2	✅

L3 = {Bread,Milk,Diaper}, {Bread,Milk,Beer}, {Bread,Diaper,Beer}, {Milk,Diaper,Beer}

Step 4: Association Rules from L2 (key rules)

From {Bread, Milk} (Support = 3/5 = 60%):

- Bread → Milk: 60/80 = **75%** ✅
- Milk → Bread: 60/80 = **75%** ✅

From {Diaper, Beer} (Support = 3/5 = 60%):

- Diaper → Beer: 60/80 = **75%** ✅
- Beer → Diaper: 60/60 = **100%** ✅

Final Strong Association Rules:

- Bread → Milk (75%) ✅
- Milk → Bread (75%) ✅
- Diaper → Beer (75%) ✅
- Beer → Diaper (100%) ✅

Summary

This question bank covers all major topics in Unit III with complete coverage:

- Market Basket Analysis — definition, real-life applications
- Support, Confidence, Lift — formulas, examples, interpretation
- Frequent Itemsets — definition, Apriori property
- Closed Itemsets and Maximal Itemsets — comparison
- Apriori Algorithm — complete step-by-step with example, advantages/disadvantages
- Generating Association Rules from Frequent Itemsets
- Improving Efficiency of Apriori — Hash-based, Partitioning, Sampling, Transaction Reduction, DIC
- FP-Growth Algorithm — definition, FP-Tree construction, complete example
- Apriori vs FP-Growth — detailed comparison table
- Types of Association Rules — single/multi-dimensional, multi-level, quantitative
- Full solved Apriori problem (PYQ-style)

Each answer is structured for a 5-mark exam response. Simple, student-friendly language throughout. Practical examples and tables included wherever possible.