CISC 7510X Final Exam

For the below questions, use the following schema definition.

```
customers(cid, fname, lname, email)
accounts(aid, cid, accttype)
securities(sid, ticker, cname, stype)
accountevents(eid, aid, sid, etype, qty, px, cashamt, ets)
```

This schema models a simplified brokerage system. Customers own one or more brokerage accounts, accounts interact with tradable securities, and all financial activity is recorded in a single accountevents ledger table.

- The customers table stores client identity information using fields such as cid, fname, lname, and email.
- The accounts table stores brokerage account metadata, including aid, the owning cid, and accttype.
- The securities table stores tradable instruments using sid, ticker, cname, and stype.
- The accountevents table is the core transactional ledger and records all account activity. Important fields include
  - aid identifying the affected account,
  - sid identifying the traded security,
  - etype describing the event type such as:
    - DEPOSIT — cash added to account ($cashamt > 0$, no sid)
    - WITHDRAW — cash removed from account ($cashamt < 0$, no sid)
    - BUY — purchase of security (sid, qty, px, $cashamt < 0$)
    - SELL — sale of security (sid, qty, px, $cashamt > 0$)
  - qty and px describing traded shares and execution price,
  - cashamt recording the resulting cash movement,
  - ets storing the event timestamp,

Pick the best answer that fits the question. Not all of the answers may be correct. If none of the answers fit, write your own answer.

1. Which attribute is the primary key of the `customers` table?
   (a) cid
   (b) aid
   (c) sid
   (d) eid

2. Which table stores brokerage account information?

   (a) customers

   (b) accounts

   (c) securities

   (d) accountevents

3. What is the result of joining `accounts` and `customers` on cid?

   (a) A list of all securities owned by customers

   (b) A mapping of accounts to their owning customers

   (c) A list of all trade events

   (d) A list of account balances

4. Which join correctly connects accounts to their events?

   (a) accounts JOIN customers ON accounts.cid = customers.cid

   (b) securities JOIN accounts ON securities.sid = accounts.sid

   (c) customers JOIN securities ON customers.cid = securities.sid

   (d) accounts JOIN accountevents ON accounts.aid = accountevents.aid

5. Which query returns all securities traded in the system?

   (a) SELECT DISTINCT sid FROM accountevents

   (b) SELECT cid FROM customers

   (c) SELECT aid FROM accounts

   (d) SELECT etype FROM accounts

6. Which query correctly joins accounts with customers?

   (a) SELECT * FROM accounts JOIN securities ON accounts.sid = securities.sid

   (b) SELECT * FROM customers JOIN accountevents ON customers.cid = accountevents.sid

   (c) SELECT * FROM accounts JOIN customers ON accounts.cid = customers.cid

   (d) SELECT * FROM accounts JOIN accountevents ON accounts.cid = accountevents.cid

7. What happens when a JOIN condition is omitted?

   (a) It becomes an INNER JOIN

   (b) It becomes a LEFT JOIN

   (c) It returns no results

   (d) It becomes a CROSS JOIN

8. What does the query `SELECT SUM(cashamt) FROM accountevents WHERE etype = 'DEPOSIT'` compute?

   (a) Number of deposit events

   (b) Total cash deposited into all accounts

   (c) Average deposit size

   (d) Largest deposit event

9. What is the role of the HAVING clause?

   (a) Filters rows before grouping

   (b) Sorts aggregated results

   (c) Joins tables after grouping

   (d) Filters groups after aggregation

10. SQL to get total cash deposited per account?

    (a) SELECT aid, SUM(cashamt) FROM accountevents WHERE etype = 'DEPOSIT' GROUP BY aid

    (b) SELECT aid, cashamt FROM accountevents GROUP BY aid

    (c) SELECT SUM(aid) FROM accountevents WHERE etype = DEPOSIT

    (d) SELECT aid, SUM(etype) FROM accountevents GROUP BY cashamt

11. SQL to join accounts with events and count events per account?

    (a) SELECT a.aid COUNT(*) FROM accounts a accountevents e WHERE a.aid = e.aid

    (b) SELECT aid, COUNT FROM accounts JOIN accountevents

    (c) SELECT a.aid, COUNT(*) FROM accounts a JOIN accountevents e ON a.aid = e.aid GROUP BY a.aid

    (d) SELECT a.aid, COUNT(e) FROM accounts a GROUP BY e.aid

12. SQL to get average BUY price per security?

    (a) SELECT sid, AVG(qty) FROM accountevents GROUP BY aid

    (b) SELECT AVG(sid), px FROM accountevents

    (c) SELECT sid, SUM(px)/COUNT(qty) FROM accounts GROUP BY sid

    (d) SELECT sid, AVG(px) FROM accountevents WHERE etype = 'BUY' GROUP BY sid

13. SQL to get net shares held per security per account?

    (a) SELECT aid, sid, SUM(qty) FROM accountevents WHERE etype = 'BUY' GROUP BY aid, sid

    (b) SELECT aid, sid, SUM(CASE WHEN etype='BUY' THEN qty ELSE -qty END) FROM accountevents GROUP BY aid, sid

    (c) SELECT aid, sid, COUNT(qty) FROM accountevents GROUP BY aid

    (d) SELECT aid, SUM(px) FROM accountevents GROUP BY sid

14. What does the following query compute? `SELECT aid, SUM(cashamt) FROM accountevents GROUP BY aid`

    (a) Total number of trades per account

    (b) Net cash position per account

    (c) Total securities per account

    (d) Average trade price per account

15. Compute net cash balance per account using accountevents, where DEPOSIT and INTEREST are positive contributions and WITHDRAW and FEE are negative contributions.

    (a) WITH cte AS (SELECT aid, CASE WHEN etype IN ('DEPOSIT','INTEREST') THEN cashamt ELSE -cashamt END AS adj FROM accountevents) SELECT aid, SUM(adj) FROM cte GROUP BY aid

    (b) SELECT aid, SUM(cashamt) FROM accounts GROUP BY aid

    (c) WITH cte AS (SELECT aid, cashamt FROM accounts) SELECT SUM(aid) FROM cte GROUP BY cashamt

    (d) SELECT aid FROM accountevents WHERE SUM(cashamt) > 0

16. Compute total BUY quantity per security, and return only securities with more than 1000 shares traded.

    (a) SELECT sid, qty FROM accountevents WHERE SUM(qty) > 1000

    (b) WITH buys AS (SELECT aid FROM accounts) SELECT sid FROM buys GROUP BY sid

    (c) SELECT sid FROM securities WHERE qty > 1000

    (d) WITH buys AS (SELECT sid, qty FROM accountevents WHERE etype = 'BUY') SELECT sid, SUM(qty) FROM buys GROUP BY sid HAVING SUM(qty) > 1000

17. Compute per-account net shares per security, where BUY adds and SELL subtracts, return only positions where net shares are non-zero.

    (a) SELECT aid, sid, SUM(qty) FROM accountevents GROUP BY aid

    (b) WITH pos AS (SELECT aid, sid, SUM(CASE WHEN etype = 'BUY' THEN qty ELSE -qty END) AS net_qty FROM accountevents GROUP BY aid, sid) SELECT * FROM pos WHERE net_qty <> 0

    (c) WITH pos AS (SELECT aid FROM accounts) SELECT sid FROM pos WHERE qty <> 0

    (d) SELECT aid, sid FROM securities GROUP BY qty

18. SQL to get a running total of cash flow per account over time.

    (a) SELECT aid, SUM(cashamt) FROM accountevents GROUP BY aid

    (b) SELECT aid, ets, SUM(cashamt) FROM accountevents GROUP BY ets

    (c) SELECT aid, ets, cashamt, SUM(cashamt) OVER (PARTITION BY aid ORDER BY ets) AS running_cash FROM accountevents

    (d) SELECT aid, cashamt FROM accountevents WHERE SUM(cashamt) OVER aid

19. Consider the query pattern: frequently retrieve all accountevents for a given account and security, ordered by event timestamp, to compute positions and running P&L. Which indexing strategy is most appropriate to optimize this workload?

    (a) Create separate single-column indexes on aid, sid, and ets, since each column is filtered independently in different queries, and rely on the query planner to combine index scans at runtime.

    (b) Create a composite index on (ets, sid, aid) because ordering by timestamp is the most common operation, and filtering by account is secondary in most analytical workloads.

    (c) Create a composite index on (aid, sid, ets) because queries filter first by account, then security, and finally require ordered traversal by event time within each account-security pair.

    (d) Avoid indexes entirely on accountevents because event tables are append-only and indexing will always degrade write performance more than it improves read performance in all financial systems.

20. The accountevents table is expected to grow to billions of rows over time. The most common query pattern retrieves all events for a given account and security ordered by timestamp for P&L reconstruction. Which physical storage strategy is most appropriate to optimize this workload?

    (a) Store accountevents as a heap table with no indexes, because sequential scans are faster than indexed lookups when the table is large and append-only workloads benefit from avoiding index maintenance overhead entirely.

    (b) Create a clustered index on (ets) only, since time-based ordering is the dominant access pattern, and rely on runtime filtering by aid and sid after the scan completes.

    (c) Partition accountevents by etype only, because separating BUY, SELL, DEPOSIT, and WITHDRAW events ensures balanced partitions and improves aggregate query performance across all accounts uniformly.

    (d) Partition accountevents by (aid, sid), and within each partition maintain ordering by ets, so that queries for a single account-security pair can be satisfied using partition pruning and ordered sequential access.