EXAM: ALL CLASSES PRIOR TO THE EXAM
- WITH AN EMPHASIS ON TOPICS DISCUSSED AFTER THE LAST EXAM
May include - entity and referential integrity constraints - data types

Even in the 2-tier and 3-tier architectures
COMPARE DB TRIGGER APPROACHES WITH THOSE WITHOUT SUCH.

**TRIGGER**

+ MAINTAIN CONSISTENCY OF DATA VALUES
+ IMPROVES PERFORMANCE BY PLACING COMPUTATION CLOSE TO THE DATA (REDUCES NEED FOR COMMUNICATION BETWEEN L AND D TIER)
+ CONCEIVABLY IF DB IMPLEMENTATION IS BETTER SUITED FOR L THEN GOOD
- SPLITS LOGIC ACROSS TWO ARCH COMPONENTS (MAKES THE TEAMWORK MORE COMPLICATED FOR DEV & MAINTENANCE)

+ DBA MAINTAINS STORED PROCEDURES
- DBA MAINTAINS STORED PROCEDURES
- DIFFICULTY IN PORTING BECAUSE OF PROPRIETARY LANGUAGES
- NEED FOR S专门LY TRAINED STAFF
How About Database Triggers?

- **Pros:** essential for achieving high efficiency
  - Reduce network load and materializing and serializing costs
  - Leave the heavy logic in the database, under the care of the DBA

- **Cons:** rarely port well across vendors
  - Difficult to introduce and manage because of DBA control
  - Business rules are context-sensitive and cannot always be applied regardless of how the data is modified
Implementational Architecture: 1

Centered on a Web server that
- Supports HTTP operations
- Usually multithreaded
Implementational Architecture: 2

Application server

- Mediates interactions between browsers and backend databases: runs computations, invoking DB transactions as needed
- Provides a venue for the business logic
- Different approaches (CGI, server scripts, servlets, Enterprise JavaBeans)

- **Tradeoffs**
  - Overhead of processes vs. threads
  - Scalability
  - Security
  - Ease of maintaining a session
Implementational Architecture: 3

Database Servers
- Hold the data, ensuring its integrity
- Manage transactions, providing
  - Concurrency control
  - Recovery

Transaction monitors can manage transactions across database systems, but within the same administrative domain.
Data Center Architecture

- Demilitarized zone (DMZ)
  - External router
  - Load balancer
- Firewall: only the router can contact the internal network
  - Internal network
  - Web servers
  - Application servers
  - Database servers
- ENHANCES SECURITY
- SCALABILITY (BY ADDING OR REMOVING RESOURCES AS NEEDED)
Application Servers

Architectural abstraction separating business logic from infrastructure

- Load balancing
- Distribution and clustering
- Availability
- Logging and auditing
- Connection (and resource) pooling
- Security

Separate programming from administration roles
Components with routine, reusable functionality

- Abstracted from the application logic or the backend systems
- Any functionality that is being repeated is a candidate for being factored out into middleware
- Enables plugging in endpoints (e.g., clients and servers) according to the stated protocols
- Often preloaded on an application server
- Simplify programmer’s task and enable refinements and optimizations
Middleware: 2

Software components that implement architectural interfaces, e.g., transaction, persistence, ...  

- **Explicit:**
  - Invoke specialized APIs explicitly
  - Difficult to create, maintain, and port applications
  - Ties application code to the middleware interface

- **Implicit:**
  - "Container" invokes the appropriate APIs
  - Based on declarative specifications
  - Relies on request interceptions or reflection
Containers

- Architectural abstraction geared for hosting business components
  - Remote method invocation
  - Threading
  - Messaging
  - Transactions

- Implementations for JEE and .NET
.NET Technology

- CORBA Client
- ActiveX Control in Browser
- Applications
- Web Browser
  - XML, HTML, HTTP (SSL)
- IIS/ASP

- Shared Property Manager
- COM+ COMPONENT
- Babylon Integration Server

- Microsoft Transaction Server
  - Microsoft Message Queue
  - Active Directory
  - Windows Operating System

- ADO, OCEDB, ODBC
- Relational DBMS
- Legacy System
Message-Oriented Middleware: 1

MoM Messaging Middleware

Asynchrony (unlike phone, like email)

- **Queues**: point to point, support posting and reading messages

- **Topics**: logical multicasts, support publishing and subscribing to application-specific topics; thus more flexible than queues

- Can offer reliability guarantees of delivery or failure notification to sender
  - Analogous to store and forward networks

- Some messages correspond to event notifications
Message-Oriented Middleware: 2

- Varies in reliability guarantees
- Usually implemented over databases
- Can be used through an invocation-based interface (i.e., registered callbacks)
Message-Driven Beans

A standardized receiver for messages

- Clients can’t invoke them directly; must send messages to them
- No need for specialized interfaces, such as `home`, `remote`, ... `vs Entity Beans`
- Easy interface to implement: mainly `onMessage()`, but limited message typing
- Stateless: thus no conversations

Callback
Methods for Message-Driven Beans

- `onMessage()`: define what actions to take when a message arrives on the destination this bean is watching