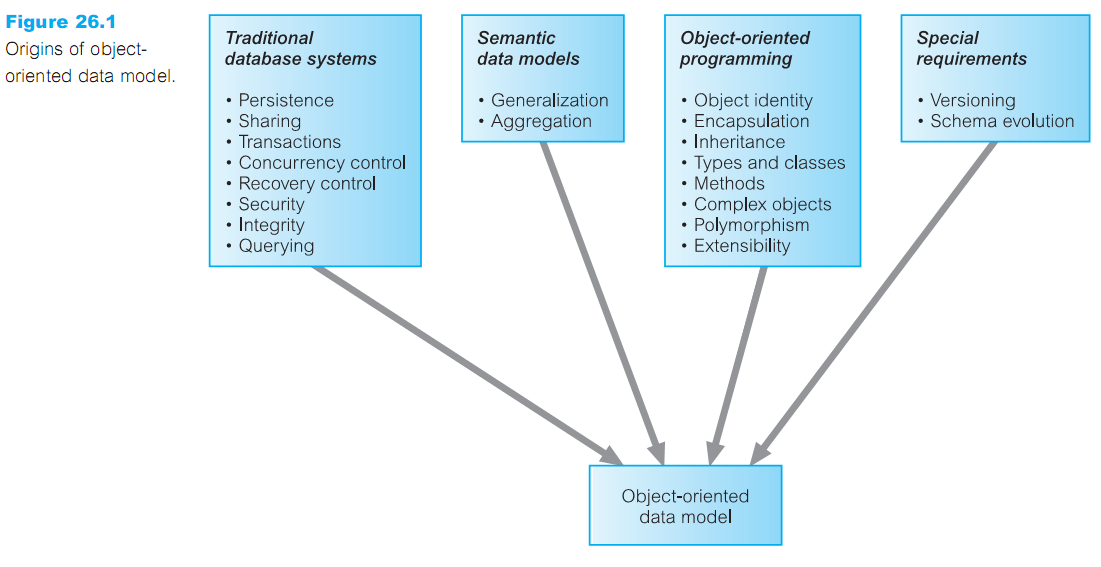
**OODB: Ch. (26) Object-Oriented DBMSs - Concepts**

1. **Compare and contrast the different deﬁnitions of object-oriented data models.**



**ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ**

**2- Describe the main modeling component of the functional data model (FDM).**

* Functional Data Model (FDM) is one of the simplest in the family of semantic data models.
* The main modeling components are **entities** and **functional relationships**.
* **Entities:**
* Entities types correspond to classes of ‘real world’ objects and are declared as   
  functions with zero arguments that return the type ENTITY. **EX: Staff() ---› ENTITY**
* Printable entity types are like to the base types in a programming language  
  and include: INTEGER, CHARACTER, STRING, REAL, and DATE. An attribute is   
  deﬁned as a functional relationship  
   **staff No(Staff) ---› STRING**

**Sex (Staff) ---› CHAR**

* We can declare a composite attribute by ﬁrst declaring the attribute to be an entity type and then declaring its components as functional relationships of the  
   entity type.

**Name () ---› ENTITY**

**Name (Staff) ---› NAME**

**FName (Name) ---› STRING**

**LName (Name) ---› STRING**

* **Relationships:**
* Functions with arguments model not only the properties (attributes) of entity types but also relationships between entity types.
* Each relationship may have an inverse relationship deﬁned.

**Manages (Staff) ---›› PropertyForRent**

**ManagedBy (PropertyForRent) ---› Staff INVERSE OF Manages**

Note: the double-headed arrow (**---››**) is used to represent a one-to-many relationship   
& Many-to-many relationships can be modeled by using the double-headed arrow in both directions.

* The FDM also supports multi-valued functions:

**ViewDate (Client, PropertyForRent) ---› DATE**

كدا اجابة السؤال خلصت ,,,, اللي جاي باقي شرح ال (FDM) في الكتاب

* **Inheritance and path expressions:**
* The FDM supports inheritance through entity types also supports the principle of substitutability. for Example:

**Staff () ---› ENTITY**

**Supervisor () ---› ENTITY**

**IS-A-STAFF (Supervisor) ---› Staff**

* The FDM allows derived (يستنتج) functions to be deﬁned from the composition of multiple functions.

**FName (Staff) ---› fName (Name (Staff))**

* This composition is called a **path expression** and may be more recognizable written in dot notation

FName (Name (Staff)) convert to Staff.Name.fName

* **Functional query languages**:
* Path expressions are also used within a functional query language.
* simple example to illustrate the language:  
   **RETRIEVE LName(Name(Staff))**

**WHERE staffNo (Staff) = ‘100’**

to retrieve the surnames of clients who staff member is "100"  
-- the equivalent dot notation may be more recognizable:

**RETRIEVE Staff.Name.lName**

**WHERE Staff.staffNo = ‘100’**

* **advantages of the FDM**
* Support for some object-oriented concepts such as Inheritance, object

identity, and navigational access.

* Support for referential integrity (التكامل المرجعي): The FDM is an entity-based data model and implicitly (ضمنيا) supports referential integrity.
* Irreducibility (عدم قابلية الاختزال): The FDM is composed of a small number of simple concepts that represent semantically irreducible units of information.
* Easy extensibility: Entity classes and functions can be added/deleted without requiring modiﬁcation to existing schema objects.
* Suitability for schema integration : FDM can be used to represent a number of different data models including relational, network, hierarchical, and object-oriented. This makes the FDM a suitable model for the integration of different schemas within multidatabase systems (MDBSs).
* Declarative query language: The query language is declarative with well understood semantics. This makes the language easy to transform and optimize.

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **What is a persistent programming language and how does it differ from an OODBMS?**

* **Persistent programming language:** A language that provides its users with the ability to preserve data across successive executions of a program, and even allows such data to be used by many different programs.
* Data in a persistent programming language is independent of any program, able to exist beyond the execution and lifetime of the code that created it.
* **How does it differ from an OODBMS?**
* **Database programming language:** A language that integrates some ideas from the database programming model with traditional programming language features.
* In contrast, a database programming language is distinguished from a persistent programming language by its integration of features beyond persistence, such as transaction management, concurrency control, and recovery.

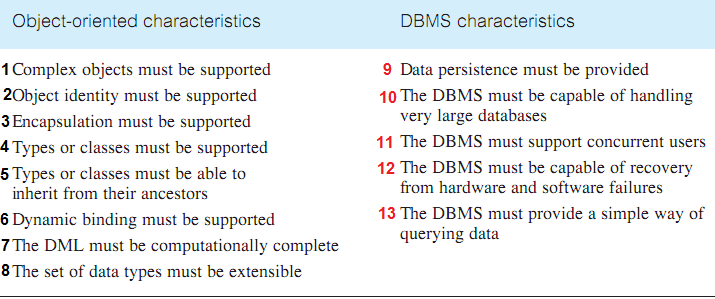
**------------------------------------------------------------------------- كدا اجابة السؤال خلصت , ---------------------------------------------------------------------------------------**

* **Persistent programming language Aims:**
* Persistent programming languages attempt to eliminate the impedance mismatch by extending the programming language with database capabilities.
* The second important aim of a persistent programming language is to maintain the same data representation in the application memory space as in the persistent store on secondary storage.
* The addition of persistence into a programming language is an important enhancement to an interactive development environment

**The Object-Oriented Database System (OODBMS) Manifesto**

It describes the main characteristics and features that a system must have to describe as an object-oriented database system.

* 13 mandatory features based on two criteria: it should be an object-oriented system and it should be a DBMS.



1. **Complex objects must be supported:**

* Complex object formed from simpler ones by constructors.
* Constructor in a class: a sequence of program instructions used to create an object.
* The simplest objects are objects such as integers, characters, strings, Booleans.
* The complex objects constructors such as tuples, sets, bags, lists, and arrays.
* The object constructors must be orthogonal: any constructor should apply to any object For example, we should be able to use not only SET(TUPLE()) and LIST(TUPLE()) but also TUPLE(SET()) and TUPLE(LIST()).

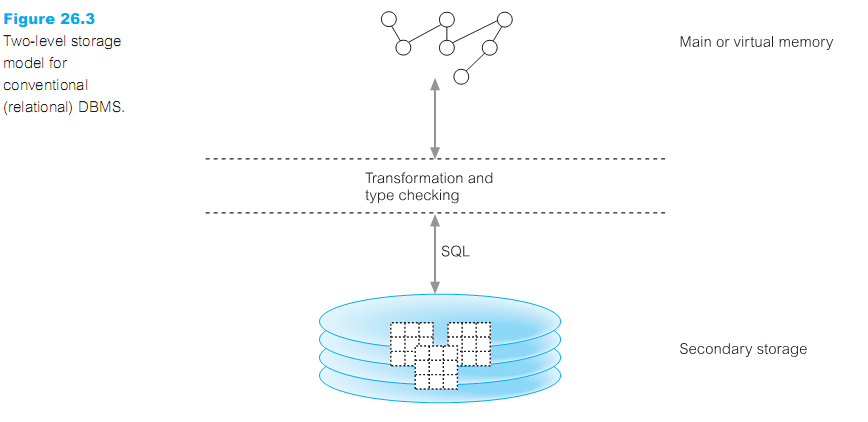
1. **Object identity must be supported:** All objects must have a unique identity that is independent of its attribute values.
2. **Encapsulation must be supported:** In an OODBMS, proper encapsulation is achieved by ensuring that programmers have access only to the interface speciﬁcation of methods, and the data and implementation of these methods are hidden in the objects.
3. **Types or classes must be supported:** The manifesto requires support for only one of these concepts. The database schema in an object-oriented system comprises(consists) a set of classes or a set of types.
4. **Types or classes must be able to inherit from their ancestors:** A subtype or subclass should inherit attributes and methods from its super type or super-class.
5. **Dynamic binding must be supported:** the system cannot bind operation names to programs at compile time. Therefore, operation names must be resolved at run-time. This delayed translation is called "late binding".
6. **The DML must be computationally complete:** In other words, the Data Manipulation Language (DML) of the OODBMS should be a general-purpose programming language and it simply means that we can express any computable function.
7. **The set of data types must be extensible:** The user must be able to build new types from the set of pre-deﬁned system types. Developers can define new types according to requirements.
8. **Data persistence must be provided:** Persistence is the ability of the programmer to have her/his data survive the execution of a process.
9. **The DBMS must be capable of managing very large databases:** In a conventional DBMS, there are mechanisms to manage secondary storage efficiently, such as indexes and buffers. An OODBMS should have similar mechanisms that are invisible to the user.
10. **The DBMS must support concurrent users:** An OODBMS should provide concurrency control mechanisms (ACID) similar to those in conventional systems.
11. **The DBMS must be capable of recovery from hardware and software failures:** An OODBMS should provide recovery mechanisms similar to those in conventional systems.
12. **The DBMS must provide a simple way of querying data:** An OODBMS must provide an ad hoc query facility that is high-level and efficient.

**Alternative Strategies for Developing an OODBMS**

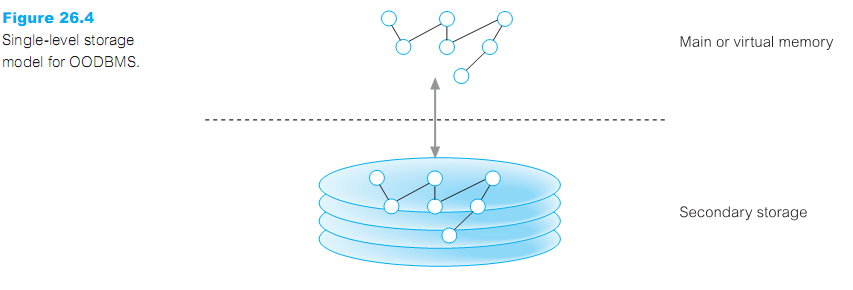
* Extend an existing object-oriented programming language with database capabilities.
* Provide extensible object-oriented DBMS libraries: extending the language, class libraries are provided that support persistence, aggregation, data types, transactions, concurrency, security, and so on.
* Embed object-oriented database language constructs in a conventional host language
* Extend an existing database language with object-oriented capabilities.

Develop a novel database data model /data language.

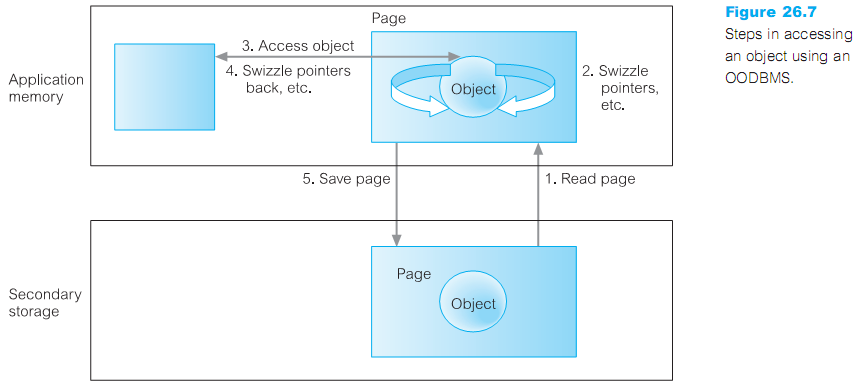
**4. Discuss the difference between the two-level storage model used by conventional DBMSs and the single-level storage model used by OODBMSs.**

Conventional DBMSs have a two-level storage model: the application storage model in main or virtual memory, and the database storage model on disk.  


* An OODBMS tries to give the illusion of a single-level storage model, with a similar representation in both memory and in the database stored on disk.



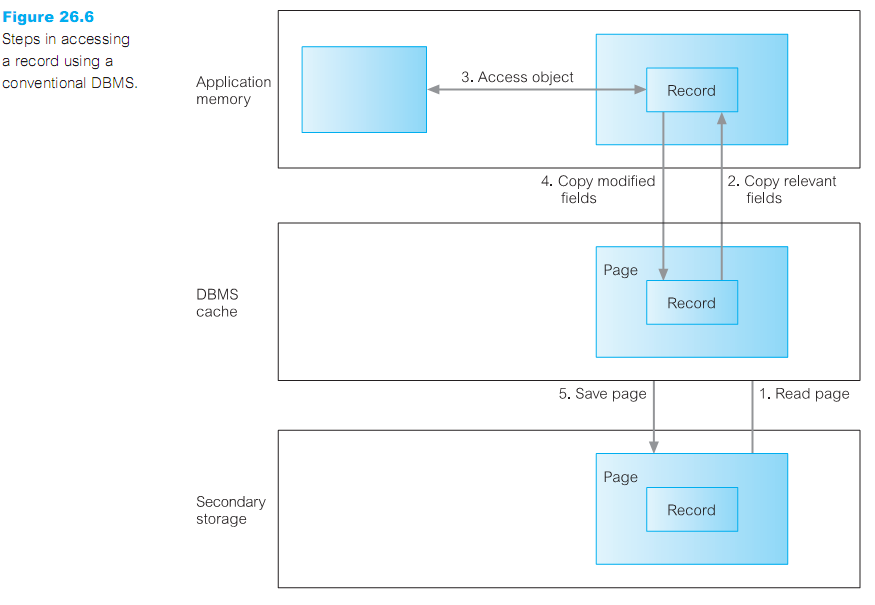
ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **How does this single-level storage model affect data access?**

* The OODBMS determines the page on secondary storage that contains the required

object using its OID.

* The OODBMS then reads that page from secondary storage and copies it into the application’s page cache within its memory space.
* The OODBMS may then carry out a number of conversions, such as:
* Swizzling references (pointers) between objects.
* Adding some information to the object’s data structure.
* Modifying the data representations for data that has come from a different hardware platform or programming language.
* The application can then directly access the object and update it, as required.
* When the OODBMS needs to swap the page out of the page cache, the OODBMS may need to carry out similar conversions as listed above, before copying the page back to secondary storage.

**\*\*\* How does this two-level storage model affect data access? السؤال دا مش موجود فى الكتاب**

* The DBMS determines the page on secondary storage that contains the required record using indexes or table scans.
* The DBMS then reads that page from secondary storage and copies it into its cache.
* The DBMS subsequently transfers the required parts of the record from the cache into the application’s memory space. Conversions may be necessary to convert the SQL data types into the application’s data types.
* The application can then update the record’s ﬁelds in its own memory space.
* The application transfers the modiﬁed ﬁelds back to the DBMS cache using SQL, again requiring conversions between data types.

Finally, at a suitable point the DBMS writes the updated page of the cache back to secondary storage.

**ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ**

1. **Describe the main strategies that can be used to create persistent objects.**

A DBMS must provide support for the storage of **persistent objects**, that is, objects that survive after the user session or application program that created them has terminated. This is in contrast to **transient objects** that last only for the invocation of the program. Persistent objects are retained until they are no longer required at this point they are deleted.

* **three schemes for implementing persistence within an OODBMS:**
* Check pointing:
* Some systems implement persistence by copying all or part of a program’s address space to secondary storage. In cases where the complete address space is saved, the program can restart from the checkpoint.
* Check pointing has two main drawbacks: typically, a checkpoint can be used only by the program that created it; second, a checkpoint may contain a large amount of data that is of no use in subsequent executions.
* Serialization:
* Some systems implement persistence by copying the closure(نهاية) of a data structure to disk.
* This process is sometimes called serialization, pickling, or in a distributed computing context, marshaling.
* Serialization has two inherent problems. First, it does not preserve object identity. Second, serialization is not incremental, and so saving small changes to a large data structure is not efficient.
* Explicit paging:
* Some persistence schemes involve the application programmer explicitly ‘paging’ objects between the application heap and the persistent store.
* With the explicit paging mechanism, there are two common methods for creating/updating persistent objects: reachability-based and allocation-based.
* Reachability-based: persistence means that an object will persist if it is reachable from a persistent root object. Any object can become persistent by adding it to the reachability tree.
* Allocation-based: persistence means that an object is made persistent only if it is explicitly declared at the application program. This can be achieved in several ways, for example:

--> By class: A class is statically declared to be persistent and all  
 instances of the class are made persistent when they are created.

--> By explicit call: An object may be speciﬁed as persistent when it is   
 created or, in some cases, dynamically at runtime.

**Orthogonal Persistence:**

* An alternative mechanism for providing persistence in a programming language is known as "orthogonal persistence"
* **Based on three fundamental principles.**

1. Persistence independence: The persistence of a data object is independent of how the program manipulates that data object.
2. Data type orthogonally: All data objects should be allowed the full range of persistence whatever their type. There are no special cases where an object is not allowed to be long-lived or is not allowed to be transient.
3. Transitive persistence: The choice of how to identify and provide persistent objects at the language level is independent of the choice of data types in the language.

* **Advantages of orthogonal persistence:**
* Orthogonal persistence is more convenient for both the programmer and the system.
* Improved programmer productivity.
* Improved maintenance.
* Support for incremental evolution.
* Automatic referential integrity.
* Protection mechanisms are consistent and suitable over the whole environment.
* **disadvantages of orthogonal persistence:**
* There is some runtime expense in a system at addressing process. Orthogonal persistence promotes transparency. a system with support for sharing among concurrent processes cannot be fully transparent.

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **What is pointer swizzling (object faulting)? Describe its different approaches.**

* **Pointer swizzling:** The action of converting object identiﬁers to main memory pointers, and back again. The aim of pointer swizzling is to optimize access to objects.
* **pointer swizzling techniques:**
* No swizzling:
* The easiest implementation of faulting objects into and out of memory is not to do any swizzling at all.
* In this case, objects are faulted into memory by the underlying object manager and a handle is passed back to the application containing the object’s OID.
* The OID is used every time the object is accessed.
* This requires that the system maintain some type of lookup table so that the object’s virtual memory pointer can be located and then used to access the object.
* As the lookup table is required on each object access, this approach could be inefficient if the same object is accessed repeatedly.
* Object referencing:
* To be able to swizzle a persistent object’s OID to a virtual memory pointer, a mechanism is required to distinguish between resident and non-resident objects. Most techniques are variations of either edge marking or node marking.

Considering virtual memory as a directed graph consisting of objects as nodes and references as directed edges, edge marking marks every object pointer with a tag bit. If the bit is set, then the reference is to a virtual memory pointer; otherwise, it is still pointing to an OID and needs to be swizzled when the object it refers to is faulted into the application’s memory space. Node marking requires that all object references are immediately converted to virtual memory pointers when the object is faulted into memory. The ﬁrst approach is a software-based technique but the second approach can be implemented using software- or hardware-based techniques.

* Hardware-based schemes:
* Hardware-based swizzling uses virtual memory access protection violations (انتهاكات) to detect accesses to non-resident objects.
* These schemes use the standard virtual memory hardware to trigger the transfer of persistent data from disk to main memory.
* The hardware approach has been used in several commercial and research systems.
* The main advantage of the hardware-based approach is that accessing memory-resident persistent objects is just as efficient as accessing transient (عابر) objects because the hardware approach avoids the overhead of residency checks incurred by software approaches.
* A disadvantage of the hardware-based approach is that it makes the provision of many useful kinds of database functionality much more difficult, such as recovery and ﬂexible buffer management policies. In addition, the hardware approach limits the amount of data that can be accessed during a transaction to the size of virtual memory. This limitation could be overcome by using some form of garbage collection to reclaim memory space, although this would add overhead and complexity to the system.

**------------------------------------------------------------------------- كدا اجابة السؤال خلصت, -------------------------------------------------------------------**

* **Classiﬁcation of pointer swizzling:**
  + Pointer swizzling techniques can be classiﬁed to three dimensions:

1. Copy versus in-place swizzling.

* When faulting objects in, the data can either be copied into the application’s local object cache or it can be accessed in-place within the object manager’s page cache.

1. Eager (حريص) versus lazy swizzling.

* Eager swizzling is the swizzling of all OIDs for persistent objects on all data pages used by the application before any object can be accessed.
* Lazy swizzling swizzles pointers only when they are accessed or discovered.

1. Direct versus indirect swizzling.

* With direct swizzling, the virtual memory pointer of the referenced object is placed directly in the swizzled pointer. With indirect swizzling; the virtual memory pointer is placed in an intermediate object, which acts as a placeholder for the actual object.
* **في الشابتر اللي فات كان فية 3 مشاكل عند استخدام ال (relational DBMSs) وهما   
  (long-duration transactions و versions و schema evolution) سبحان الله ال 3 مشاكل دول اتحولوا لما استخدمنا (OODBMS)**
* **هيشرح كل واحدة علي حدي وجايب سؤال عن كل واحدة فيهم اللي هما سؤال رقم 8 , 9 , 10**

**وجب التنوية بس علشان تبقي فاهم انتا وصلت لفين؟!! وانتا بتزاكر اية؟!! علشان كل حاجة منفدة علي بعضها**

☺

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Describe the types of transaction protocol that can be useful in design applications.**
   * + - A transaction is a logical unit of work, which should always transform the database from one consistent state to another.
     + There are short duration transactions such as those found in business applications and long duration transactions involving complex objects, such as those found in engineering and design applications.

**ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ**

1. **Discuss why version management may be a useful facility for some applications.**

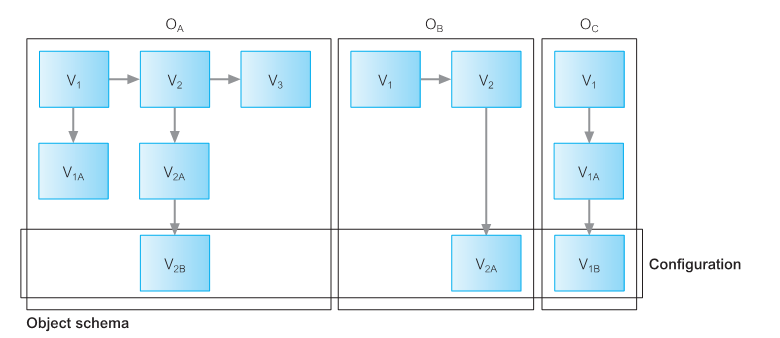
- The process of maintaining the evolution of objects is known as **version management.**

- **An object version** represents an identiﬁable state of an object.

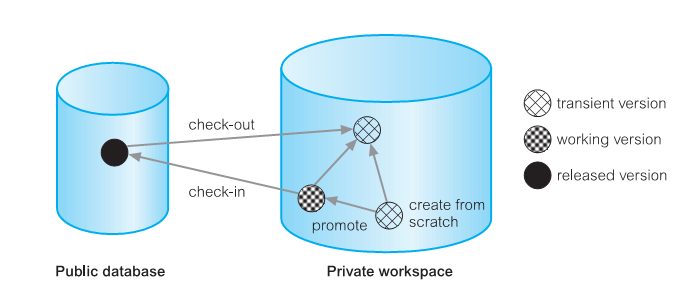
- **a version history** represents the evolution of an object.

- Versioning should allow changes to the properties of objects to be managed in such a way that object references always point to the correct version of an object.

- Example on version management for three objects: OA, OB, and OC



* For example, we can determine that object OA consists of versions V1, V2, V3;   
   V1A is derived from V1, and V2A and V2B are derived from V2.   
  This ﬁgure also shows an example of a conﬁguration of objects, consisting of V2B of OA, V2A of OB, and V1B of OC.
* **types of version ( 3 types ):**
* Transient versions: A transient version is considered unstable and can be updated and deleted. It can be created from new by checking out a released version from a public database. Transient versions are stored in the creator’s private workspace.
* Working versions: A working version is considered stable and cannot be updated, but it can be deleted by its creator. It is stored in the creator’s private workspace.
* Released versions: A released version is considered stable and cannot be updated or deleted. It is stored in a public database by checking in a working version from a private database.



كل الكلام المكتوب دة عبارة عن شرح ال (version) كامل بانواعة وبسلاطاتة   
بس السؤال عاوزك تناقش لية ال (version management) وسيلة مهمة ممكن نستخدمها في التطبيقات  
الاجابة ممكن تجيبها من جزيئة المثال انه بيعمل (determine object versions) و كمان (conﬁguration of objects) ومفيش مشكلة انك تزود الاجابة بتعريف عن ال(version) وانواعة بس المهم تكتب مضمون الاجابة **ونفس الكلام مع السؤال الجاي**

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

1. **Discuss why schema control may be a useful facility for some applications.**

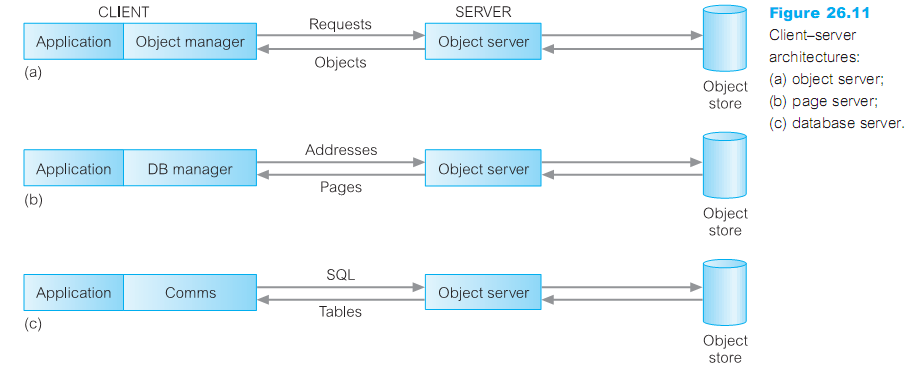
* Design is an incremental process and evolves with time. To support this process, applications require considerable ﬂexibility in dynamically deﬁning and modifying the database schema.
* Typical changes to the schema include:
  + Changes to the class deﬁnition:
    - Modifying attributes;
    - Modifying methods.
  + Changes to the inheritance hierarchy:
    - Making a class S the superclass of a class C;
    - Removing a class S from the list of superclasses of C;
    - Modifying the order of the superclasses of C.
    - (3) Changes to the set of classes, such as creating and deleting classes and  
       modifying class names.
* Rules for schema consistency, called **schema invariants**(ثوابت)

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

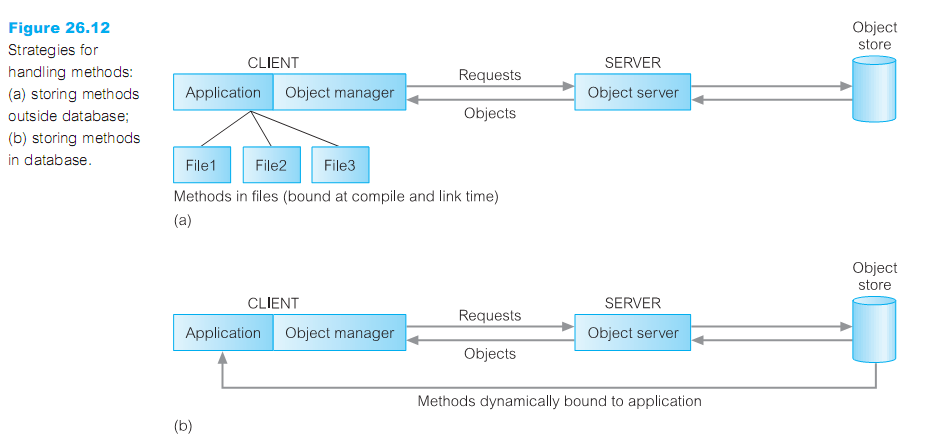
1. **Describe the different architectures for an OODBMS.**

* **Client-server**
* Many commercial OODBMSs are based on the client–server architecture to provide data to users, applications, and tools in a distributed environment.
* Not all systems use the same client–server model.
* We can distinguish three basic architectures for a client–server DBMS

(1) Object server (2) Page server (3) Database server



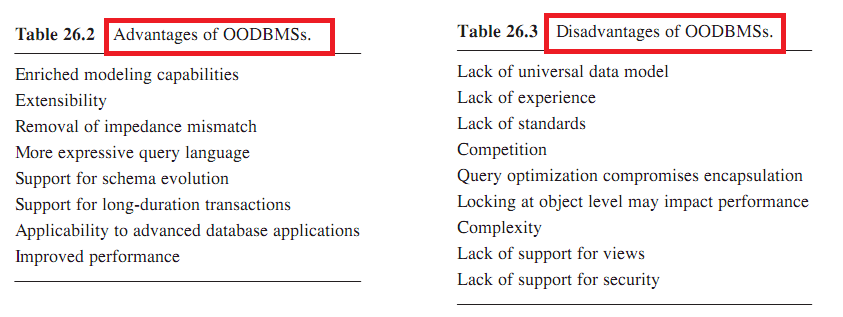
* **Storing and executing methods**
* There are two approaches to handling methods: store the methods in external ﬁles and store the methods in the database.



* **storing methods in the database have several beneﬁts**:
* It eliminates redundant (unnecessary) code.
* It simpliﬁes modiﬁcation.
* Methods are more secure.
* Methods can be shared concurrently.
* Improved integrity

**ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ**

1. **List the advantages and disadvantages of an OODBMS.**

****

* **Advantages:**

1. Enriched modeling capabilities: The object-oriented data model allows the ‘real world’ to be modeled more closely. The object, which encapsulates both state and behavior, is a more natural representation of real-world objects. An object can store all the relationships it has with other objects, including many-to-many relationships.
2. Extensibility: OODBMSs allow new data types to be built from existing types.
3. Removal of impedance mismatch: A single language interface between the Data Manipulation Language (DML) and the programming language overcomes the impedance mismatch.
4. More expressive query language: Navigational access from one object to the next is the most common form of data access in an OODBMS. Navigational access is more suitable for handling parts explosion, recursive queries, and so on.
5. Support for schema evolution: The tight coupling between data and applications in an OODBMS makes schema evolution more feasible
6. Support for long-duration transactions: Some OODBMSs use a different protocol to handle the types of long-duration transaction that are common in many advanced database applications.
7. Applicability to advanced database applications: there are many areas where traditional DBMSs have not been successful, such as, computer-aided design (CAD), computer-aided software engineering (CASE). The modeling capabilities of OODBMSs have made them suitable for these applications.
8. Improved performance: there have been a number of benchmarks that have suggested OODBMSs provide signiﬁcant performance improvements over relational DBMSs.

* **Disadvantages:**

1. Lack of universal data model: there is no universally agreed data model for an OODBMS and most models lack a theoretical foundation.
2. Lack of experience: the use of OODBMSs is still limited. This means that we do not yet have the level of experience that we have with traditional systems.
3. Lack of standards: There is a general lack of standards for OODBMSs . there is no standard object-oriented query language.
4. Lack of support for views: most OODBMSs do not provide a view mechanism.
5. Lack of support for security: OODBMSs do not provide security mechanisms.
6. Complexity: The increased functionality provided by an OODBMS make them more complex than that of traditional DBMSs.
7. Competition: Perhaps one of the most signiﬁcant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products.
8. Query optimization compromises encapsulation: Query optimization requires an understanding of the underlying implementation to access the database efﬁciently
9. Locking at object level may impact performance: Many OODBMSs use locking as the basis for a concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.

ـــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**With My Best Wishes   
 " Mohamed Noaman "**