Gambella University

College of Engineering and Technology

Department of Computer Science

Advanced Database Systems

Course Code: CoSc2042

Chapter 1

Concepts for object-oriented database

Overview

- ODBs were developed for applications that have requirements requiring **more complex** structures for stored objects.
- A key feature of object data- bases is the **power** they give the designer
- Another reason is the **vast increase** in the use of object-oriented programming languages for developing software applications.

1.1. Introduction

- The term **object-oriented**—abbreviated **OO**
- An object typically has two components:
  - **state** (value) and **behaviour** (operations).

Types of objects

- Objects in an OOPL exist only during program execution; therefore, they are called **transient objects**
- An OO database can extend the existence of objects so that they are stored permanently in a database, and hence the objects become **persistent objects**

Basic OO Concepts

Encapsulation

- Some OO models insist that all operations a user can apply to an object must be pre- defined.
- **complete encapsulation** has been relaxed in most OO data models for two reasons
  - **i.** **First**, database users often need to know the attribute names so they can specify selection conditions on the attributes to retrieve specific objects.
  - **ii.** **Second**, complete encapsulation implies that any simple retrieval requires a predefined operation, thus making adhoc queries difficult to specify on the fly.

- To encourage encapsulation, an operation is defined in two parts.

I. The first part, called the **signature** or **interface** of the operation, specifies the operation name and arguments (or parameters).

II. The second part, called the **method** or **body**, specifies the implementation of the operation, usually written in some general-purpose programming language.

Other OO concepts

type and class hierarchies and inheritance

✈This permits specification of new types or classes that inherit much of their structure and/or operations from previously defined types or classes. **operator overloading / operator polymorphism**

✈refers to an operation's ability to be applied to different types of objects.

1.2. Object Identity, and Object structure, and Type Constructors

Object Identity

- ✈ **A unique identity** is assigned to each independent object stored in the database. This unique identity is typically implemented via a unique, system-generated **object identifier (OID).**
- ✈ The value of an OID may not be visible to the external user.
- ✈ The main property required of an OID is that it be **immutable**
- ✈ It is also **inappropriate** to base the OID on the physical address of the object in storage

Complex Type Structures for Objects and Literals

- In ODBs, a complex type may be constructed from other types by nesting of type constructors.
- The three most basic constructors are
  - **atom**, **struct** (or tuple), and **collection**.

The **atom constructor** is used to represent all basic atomic values

Struct constructor can create standard structured types, such as the tuples (record types) in the basic relational model.

Collection types have individual differences among them.

- The **set constructor** will create objects or literals that are a set of distinct elements {i1, i2, … .. in}, all of the same type.
- The **bag constructor** (also called a multiset) is similar to a set except that the elements in a bag need not be distinct.
- The **list constructor** will create an ordered list [i1, i2, … , in] of OIDs or values of the same type. A list is similar to a bag except that the elements in a list are ordered, and hence we can refer to the first, second, or jth element.

Collection types…..

- The **array constructor** creates a singledimensional array of elements of the same type. The main **difference** between array and list is that a list can have an arbitrary number of elements whereas an array typically has a maximum size.
- Finally, the **dictionary constructor** creates a collection of key-value pairs (K, V), where the value of a key K can be used to retrieve the corresponding value V.

```
define type EMPLOYEE
    tuple ( Fname:        string;
            Minit :       char;
            Lname:        string;
            Ssn:          string;
            Birth_date:   DATE;
            Address:      string;
            Sex:          char;
            Salary:       float;
            Supervisor:   EMPLOYEE;
            Dept:         DEPARTMENT;
define type DATE
    tuple ( Year:         integer;
            Month:        integer;
            Day:          integer; );
define type DEPARTMENT
    tuple ( Dname:        string;
            Dnumber:      integer;
            Mgr:          tuple ( Manager:     EMPLOYEE;
                                  Start_date:  DATE; );
            Locations:    set(string);
            Employees:    set(EMPLOYEE);
            Projects:     set(PROJECT); );
```

Encapsulation of Operations, Methods and Persistence of Objects

Encapsulation

- The concept of encapsulation is applied to database objects in ODBs by defining the behaviour of a type of object based on the operations that can be externally applied to objects of that type.
- For database applications, the requirement that all objects be completely encapsulated is **too stringent** .
  One way to relax this requirement is to divide the structure of an object into visible and hidden attributes (instance variables).

Visible attributes can be seen by and are directly accessible to the database users and programmers via the query language.

The **hidden attributes** of an object are completely encapsulated and can be accessed only through predefined operations.

- The term class is often used to refer to a type definition, along with the definitions of the operations for that type.

```
define class EMPLOYEE
    type tuple (   Fname:          string;
                   Minit:          char;
                   Lname:          string;
                   Ssn:            string;
                   Birth_date:     DATE;
                   Address:        string;
                   Sex:            char;
                   Salary:         float;
                   Supervisor:     EMPLOYEE;
                   Dept:           DEPARTMENT; );
    operations     age:            integer;
                   create_emp:     EMPLOYEE;
                   destroy_emp:    boolean;
end EMPLOYEE;
define class DEPARTMENT
    type tuple (   Dname:          string;
                   Dnumber:        integer;
                   Mgr:            tuple ( Manager:      EMPLOYEE;
                                           Start_date:   DATE; );
                   Locations:      set (string);
                   Employees:      set (EMPLOYEE);
                   Projects        set(PROJECT); );
    operations     no_of_emps:     integer;
                   create_dept:    DEPARTMENT;
                   destroy_dept:   boolean;
                   assign_emp(e: EMPLOYEE): boolean;
                   (* adds an employee to the department *)
                   remove_emp(e: EMPLOYEE): boolean;
                   (* removes an employee from the department *)
end DEPARTMENT;
```

Continued…..

- Typical operations include the **object constructor** operation (often called new), which is used to create a new object, and the **destructor** operation, which is used to destroy (delete) an object. A number of **object modifier** operations can also be declared to modify the states (values) of various attributes of an object.
- An operation is typically applied to an object by using the **dot notation**. For example, if d is a reference to a DEPARTMENT object, we can invoke an operation such as no_of_emps by writing d.no_of_emps. Similarly, by writing d.destroy_dept, the object referenced by d is destroyed (deleted). The only exception is the constructor operation, which returns a reference to a new DEPARTMENT object.

Specifying Object Persistence via Naming and Reachability

- **Transient objects** exist in the executing program and disappear once the program terminates.
- **Persistent objects** are stored in the database and persist after program termination.
- The typical mechanisms for making an object persistent are naming and reachability.

- The **naming mechanism** involves giving an object a unique persistent name within a particular database.
  This persistent object name can be given via a specific statement or operation in the program

```
define class DEPARTMENT_SET
    type set (DEPARTMENT);
    operations add_dept(d: DEPARTMENT):  boolean;
        (* adds a department to the DEPARTMENT_SET object *)
            remove_dept(d: DEPARTMENT): boolean;
        (* removes a department from the DEPARTMENT_SET object *)
            create_dept_set:      DEPARTMENT_SET;
            destroy_dept_set:     boolean;
end Department_Set;
...
persistent name ALL_DEPARTMENTS: DEPARTMENT_SET;
(* ALL_DEPARTMENTS is a persistent named object of type DEPARTMENT_SET *)
...
d:= create_dept;
(* create a new DEPARTMENT object in the variable d *)
...
b:= ALL_DEPARTMENTS.add_dept(d);
(* make d persistent by adding it to the persistent set ALL_DEPARTMENTS *)
```

Continued….

- The named persistent objects are used as entry points to the database through which users and applications can start their database access.
- The **reachability mechanism** works by making the object reachable from some other persistent object. An object B is said to be reachable from an object A if a sequence of references in the database lead from object A to object B.
- For example, we can define a class DEPARTMENT_SET whose objects are of type set(DEPARTMENT). We can create an object of type DEPARTMENT_SET, and give it a persistent name ALL_DEPARTMENTS, as shown in Figure 1.3. Any DEPARTMENT object that is added to the set of ALL_DEPARTMENTS by using the add_dept operation becomes persistent by virtue of its being reachable from ALL_DEPARTMENTS.

Notice the difference between traditional database models and ODBs in this respect.

Type Hierarchies and Inheritance

- **Inheritance** allows the definition of new types based on other predefined types, leading to a **type** (or **class**) **hierarchy**.
- A **type** is defined by assigning it a type **name** and then defining a **number** of attributes (instance variables) and operations (methods) for the type.

TYPE_NAME: function, function, … , function e.g

PERSON: Name, Address, Birth_date, Age, Ssn

Continued….

- The **subtype** then inherits all the functions of the predefined type, which is referred to as the **supertype**

EMPLOYEE: Name, Address, Birth_date, Age, Ssn, Salary, Hire_date, Seniority

STUDENT: Name, Address, Birth_date, Age, Ssn, Major, Gpa

we can declare EMPLOYEE and STUDENT as follows:

PERSON: Name, Address, Birth_date, Age, Ssn

EMPLOYEE subtype-of PERSON: Salary, Hire_date, Seniority

STUDENT subtype-of PERSON: Major, Gpa

Constraints on Extents Corresponding to a Type Hierarchy

- An **extent** is a named persistent object whose value is a persistent collection that holds a collection of objects

of the same type that are stored permanently in the database.

- In this case, the constraint is that every object in an extent that corresponds to a subtype must also be a member of the extent that corresponds to its supertype.
- The ODMG model distinguishes between type inheritance—called inter- face inheritance and denoted by a colon (:)—and the extent inheritance constraint— denoted by the keyword EXTEND.

Other Object-Oriented Concepts

Polymorphism of Operations (Operator Overloading)

- This concept allows the same operator name or symbol to be bound to two or more different implementations of the operator, depending on the type of objects to which the operator is applied.
  - **early (**or **static) binding**
  - **late (**or **dynamic) binding**.

Multiple Inheritance and Selective Inheritance

- **Multiple inheritance** occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both supertypes.

e.g ENGINEERING_MANAGER

- There are several techniques for dealing with ambiguity in multiple inheritance.

I. One solution is to have the system check for ambiguity when the subtype is created, and to let the user explicitly choose which function is to be inherited at this time.

II. A second solution is to use some system default.

III. A third solution is to disallow multiple inheritance altogether if name ambiguity occurs, instead forcing the user to change the name of one of the functions in one of the supertypes.

Selective inheritance

- **Selective inheritance** occurs when a subtype inherits only some of the functions of a supertype. Other functions are not inherited.
- In this case, an **EXCEPT** clause may be used to list the functions in a super- type that are not to be inherited by the sub- type. The mechanism of selective inheritance is not typically provided in ODBs, but it is used more frequently in artificial intelligence applications.

Chapter 2

Query processing and optimization

Query processing :

- In general, a **query** is a form of questioning, in a line of inquiry
- A **query language** is a language in which user requests information from the database
- The aim of query processing is to find information in one or more databases and deliver it to the user **Query optimization**:

It is the process of choosing a suitable execution strategy for processing a query

- Two internal representations of a query:
  - **Query Tree**
  - **Query graph**

25

Query Processing and Optimization

- Scanner: identify language components.
  keywords, attribute, relation names
- Parser : check query system
- Validation: check attributes & relations
- Query tree (query graph) : internal representation
- Execution strategy: plan

- Query optimization : choose a strategy (reasonably efficient strategy)

Query Processing (cont…)

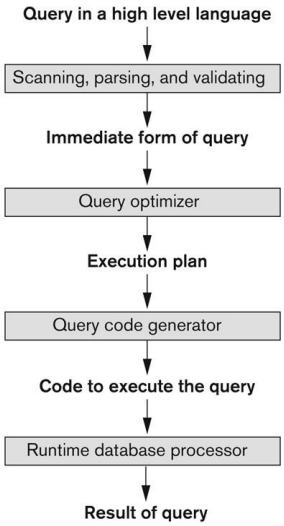


Figure 15.1
Typical steps when processing a high-level query.

Code can be:

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

27

Query Processing (cont…)

- Typical stages in query decomposition(cont…)

– **Semantic Analysis**: to reject normalized queries that are not correctly formulated or contradictory.
  - Incorrect if components do not contribute to generate result.
  - Contradictory if the predicate can not be satisfied by any tuple.
– **Simplification**: to detect redundant qualifications, eliminate common sub-expressions, and transform the query to a semantically equivalent but more easily and effectively computed form.
– **Query Restructuring** When more than one translation is possible, use transformation rules to re arranging nodes so that the most restrictive condition will be executed first.

28

Transformation Rules for Relational Algebra

- Relational algebra operations:
- Select, Project , Join, Union, Intersection, Cartesian Poduct
- refer the text book for detail understanding

✦ General Transformation rules for relational algebra

1. Cascade of S: A conjunctive selection condition can be broken up into a cascade (sequence) of individual S operations:
   - $\sigma_{c1 \text{ AND } c2 \text{ AND } \ldots \text{ AND } cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$ 2.

   Commutativity of S:
   - The S operation is commutative:
   - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$

3. Cascade of p: In a cascade (sequence) of p operations, all but the last one can be ignored:
   - $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List1}(R)$

29

Transformation Rules for Relational Algebra (cont…)

4. Commuting s with p:
   – If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:
     - $\pi_{A1, A2, ..., An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, ..., An}(R))$

5. Commuting of $\bowtie$ and X: both are commutative

   $R \bowtie_c S = S \bowtie_c R$, RXS=SXR

30

Transformation Rules for Relational Algebra (cont…)

6. Commuting p with ⋈ : If projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S and the join condition c involves only attributes in L, the two operations can be commuted as follows:

   – $\pi_L(R \bowtie_C S) = (\pi_{A1, ..., An}(R)) \bowtie_C (\pi_{B1, ..., Bm}(S))$

7. Converting a (s, x) sequence into ⋈ : If the condition c of a s that follows a x Corresponds to a join condition, convert the (s, x) sequence into a ⋈ as follows:

   $(\sigma_C(R \times S)) = (R \bowtie_C S)$

31

Transformation Rules for Relational Algebra (cont…)

- Reading Assignment

– 8. Commutativity of THETA JOIN/Cartesian Product

R X S is equivalent to S X R

Also holds for Equi-Join and Natural-Join

$(R \bowtie_{c1} S) = (S \bowtie_{c1} R)$

– 9. Commutativity of SELECTION with THETA JOIN

- If the predicate c_1 involves only attributes of one of the relations (R) being joined, then the Selection and Join operations commute

10. Commuting SELECTION with SET OPERATIONS

11. Commuting PROJECTION with UNION

32

Query Processing

- **Query Block**:
- It is the basic unit that can be translated into the algebraic operators

– A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.

– **Nested queries** within a query are identified as separate query blocks.

- Example: Find the name of employees whose salary is below the average salary of all employees who are working at department number 5

33

Query Processing (cont…)

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SALARY < C (  SELECT   AVG (SALARY)
                       FROM     EMPLOYEE
                       WHERE    DNO = 5);
```

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SALARY < C
```

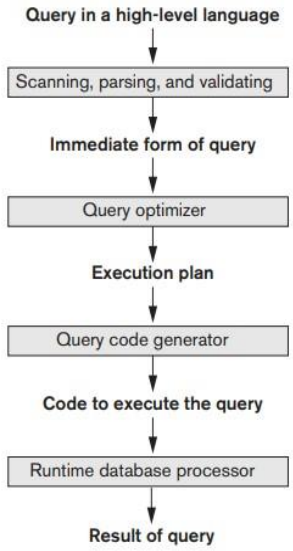
```
SELECT   AVG (SALARY)
FROM     EMPLOYEE
WHERE    DNO = 5
```

$\pi_{FNAME, LNAME}(\sigma_{SALARY<C}(EMPLOYEE))$

$\mathcal{F}_{AVGSALARY}(\sigma_{DNO=5}(EMPLOYEE))$

34

Steps processing high level query



Code can be:

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

Translating SQL Queries into Relational Algebra

- An SQL query is first translated into an equivalent extended relational algebra expression (as query tree) then optimized.
- SQL queries are decomposed into **query blocks.**
- Nested queries within a query are identified as separate query blocks.

Common notations of Relational Algebra

Operation	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Q. Find the name of employees whose salary is below the average salary of all employees who are working at department number 5?

Figure 2.2
Schema diagram for the COMPANY relational database schema.

Ex: SELECT Lname, Fname FROM EMPLOYEE WHERE Salary > (SELECT MAX (Salary) FROM EMPLOYEE WHERE Dno=5);

Solution for example

The query be decomposed into two blocks.

The inner block is:

(SELECT MAX (Salary) FROM EMPLOYEE WHERE Dno=5) | $\pi_{MAX\ Salary}(\sigma_{Dno=5}(EMPLOYEE))$ The outer query block is:

SELECT Lname, Fname FROM EMPLOYEE WHERE Salary > c | $\pi_{Lname,Fname}(\sigma_{Salary>c}(EMPLOYEE))$

2.2 Basic Algorithms for Executing Query Operations

2.2.1. Algorithms for SELECT Operation

Implementation Options for the SELECT Operation

OP1: $\sigma_{Ssn = '123456789'}$ (EMPLOYEE) equality comparison on key attribute

OP2: $\sigma_{Dnumber > 5}$ (DEPARTMENT) nonequality comparison on key attribute

OP3: $\sigma_{Dno = 5}$ (EMPLOYEE) equality comparison on non key attribute

OP4: $\sigma_{Dno = 5 \text{ AND } Salary > 30000 \text{ AND } Sex = 'F'}$ (EMPLOYEE) conjunctive condition

OP5: $\sigma_{Essn = '123456789' \text{ AND } Pno = 10}$(WORKS_ON) conjunctive condition and composite key

OP6: SELECT *FROM EMPLOYEE WHERE Dno IN (3,27, 49)

OP7: SELECT First_name, Lname FROM Employee
WHERE ((Salary*Commission_pct) + Salary) > 15000;

Search Methods for Simple Selection

- Known as **file scans**, because they scan the records of a file to search.
- If the search algorithm involves the use of an index, the index search is called an **index scan**.
- some of the search algorithms
  - **S1—Linear search (brute force algorithm)**. Retrieve every record in the file, and test whether its attribute values satisfy the selection condition.

Cont'd..

- **S2—Binary search**. If the selection condition involves an equality comparison on a key attribute on which the file is ordered. E.g **OP1**
- **S3a—Using a primary index.** If the selection condition involves an equality comparison on a key attribute with a primary index. E.g **OP1**
- **S3b—Using a hash key**. If the selection condition involves an equality comparison on a key attribute with a hash key. E.g **OP1**
- **S4—Using a primary index to retrieve multiple records.** If the comparison condition is >, >=, <, or <= on a key field with a primary index. E.g **OP2**

Cont'd

- **S5—Using a clustering index to retrieve multiple records.** If the selection condition involves an equality comparison on a non-key attribute with a clustering index **E.g OP3**
- **S6—Using a secondary (B+-tree) index on an equality comparison.** This search method can be used to retrieve a single record if the indexing field is a key (has unique values) or to retrieve multiple records if the indexing field is not a key e.g OP
  - Queries involving a range of values (e.g., 3,000 <= Salary <= 4,000) in their selection are called **range queries**.

Cont'd

- **S7a—Using a bitmap index**. If the selection condition involves a set of values for an attribute (e.g., Dnumber in (3,27,49) in OP6)
  - The corresponding bitmaps for each value can be OR-ed to give the set of record ids that qualify.
- **S7b—Using a functional index.** In OP7, the selection con- dition involves the expression
  ((Salary*Commission_pct) + Salary).

If there is a functional index defined as

```
CREATE INDEX income_ix
ON EMPLOYEE (Salary + (Salary*Commission_pct));
```

Then this index can be used to retrieve employee records that qualify.

Search Methods for Conjunctive Selection

- If a condition of a SELECT operation is a conjunctive condition. That is, if it is made up of several simple conditions connected with the **AND** logical connective, such as OP4 above.

OP4: $\sigma_{Dno= 5 \text{ AND } Salary > 30000 \text{ AND } Sex = 'F'}$ (EMPLOYEE)

Search Methods for Disjunctive Selection

- Compared to a conjunctive selection condition, a disjunctive condition (where simple conditions are connected by the OR logical connective rather than by AND) is much harder to process and optimize.
- ex OP4: $\sigma_{Dno=5 \text{ OR } Salary > 30000 \text{ OR } Sex = 'F'}(EMPLOYEE)$
- Records satisfying the disjunctive condition are the union of the records satisfying the individual conditions.

Algorithms for PROJECT and Set Operations

- A PROJECT operation $\pi_{<attribute\ list>}(R)$ from relational algebra implies that after projecting R on only the columns in the list of attributes, any duplicates are removed by treating the result strictly as a set of tuples.
- However, Q:SELECT Salary FROM EMPLOYEE produces a list of salaries of all employees.
- **Hashing** can also be used to eliminate duplicates

Set operations

- **CARTESIAN PRODUCT** operation R × S is expensive because its result includes a record for each combination of records from R and S.
  - Each record in the result includes all attributes of R and S.
  - If R has n records and j attributes, and S has m records and k attributes, the result relation for R × S will have **n * m** records and each record will have **j + k** attributes.

Use of Anti-Join for SET DIFFERENCE (or EXCEPT or MINUS in SQL)

- Suppose we want to find out which departments have no employees in the schema of Figure 2.2:

  Select Dnumber from DEPARTMENT MINUS Select Dno from EMPLOYEE; can be converted into the following:

```
SELECT DISTINCT DEPARTMENT.Dnumber FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.Dnumber A = EMPLOYEE.Dno
```

Using Heuristics in Query Optimization

- **Query optimization** is an activity conducted by a query optimizer in a DBMS to select the best available strategy for executing the query.
- One of the main heuristic rules is to apply SELECT and PROJECT operations before applying the JOIN or other binary operations
- **A query tree** is used to represent a relational algebra or extended relational algebra expression
- **A query graph** is used to represent a relational calculus expression.

Notation for Query Trees and Query Graphs

I. Query tree represents the **input** relations of the query as **leaf nodes** of the tree,
II. Represents the relational algebra **operations** as **internal nodes**.
III. The order of execution of operations starts at the leaf nodes, which represents the **input database** relations for the query, and ends at the root node, which represents the **final operation** of the query.
IV. The execution terminates when the root node operation is executed and produces the result relation for the query.

the query tree represents a specific order of operations for executing a query.

Question

- For every project located in ‘Stnafford’, retrieve the project number, the controlling department number, and the department manager’s last name, address, and birthdate.
  1. Write Algebra expression
  2. Write SQL

Answer

1. relational algebra expression:

   $\pi_{Pnumber,Dnum,Lname,Address,Bdate}(((\sigma_{Plocation='Stafford'}(PROJECT))$
   Dnum=Dnumber(DEPARTMENT)) Mgr_ssn=Ssn(EMPLOYEE))

2. This corresponds to the following SQL query:

   SELECT P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
   FROM PROJECT P, DEPARTMENT D, EMPLOYEEE
   WHERE P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND
   P.Plocation= 'Stafford';

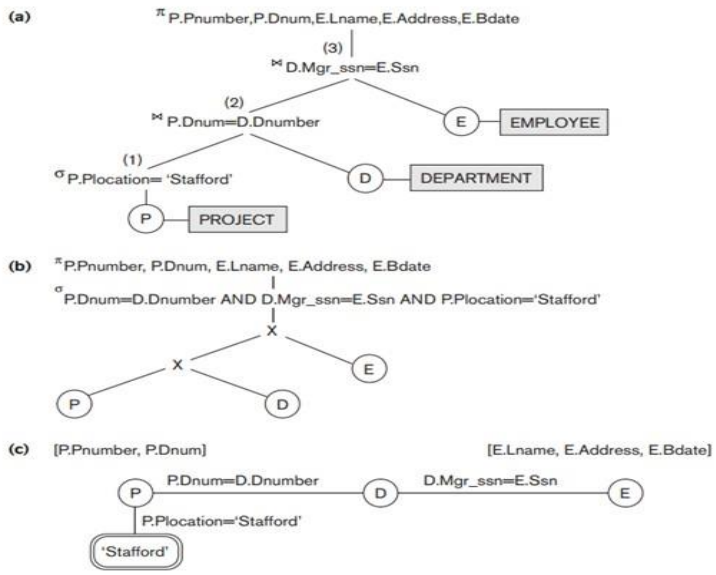


Figure
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Figure shows a query tree and graph

Query graph

✈2.3(c) shows the **query graph**.

I. **Relations** in the query are represented by relation nodes, which are displayed as **single circles**.

II. **Constant values**, typically from the query selection conditions, are represented by constant nodes, which are displayed as **double circles or ovals**.

III. **Selection and join** conditions are represented by the **graph edges**

IV. Finally, the **attributes** to be retrieved from each relation are displayed in **square brackets** above each relation.

Heuristic Optimization of Query Trees

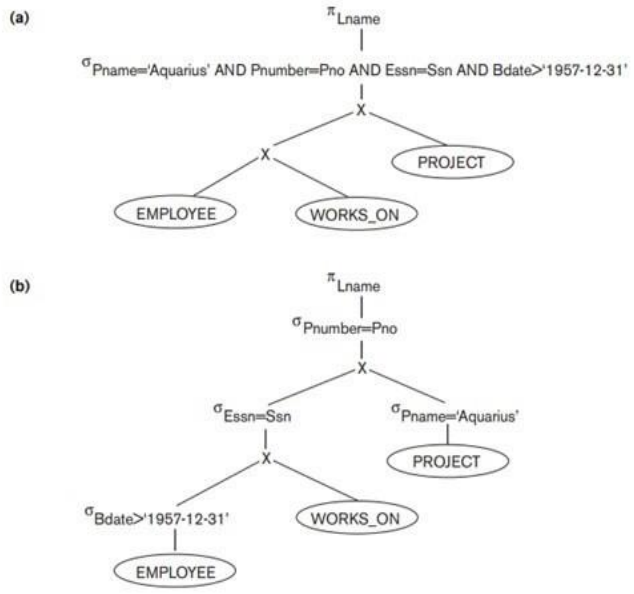
Example of Transforming a Query

Find the last names of employees born after 1957 who work on a project named ‘Aquarius’.

SELECT E.Lname FROM EMPLOYEE E, WORKS_ON W, PROJECT P

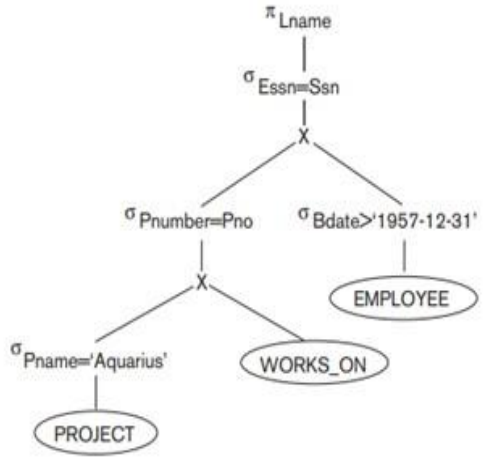
WHERE P.Pname=‘Aquarius’ AND P.Pnumber=W.Pno AND E.Essn=W.Ssn AND E.Bdate > ‘1957-12-31’;

Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q. (b) Moving SELECT operations down the query tree. (c) Applying the more restrictive SELECT operation first.



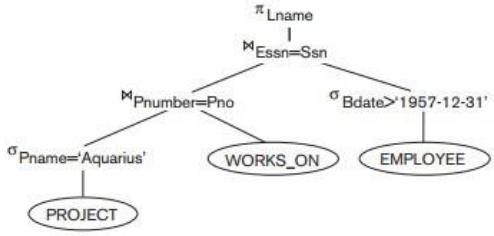
- PROJECT (π) operations reduces the attributes (columns) of the intermediate relations SELECT operations reduce the number of tuples (records).

(c)

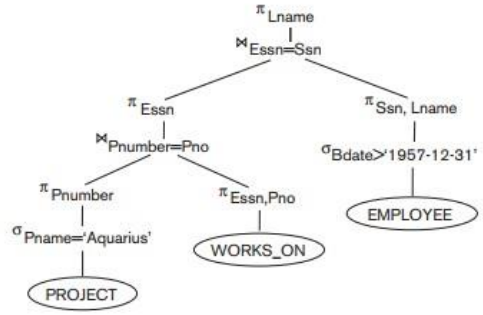


Steps in converting a query tree during heuristic optimization. (d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations. (e) Moving PROJECT operations down the query tree.

(d)



(e)



Transformation Rules for Relational Algebra Operations

- **Cascade of σ.** σ c1AND c2 AND … AND cn(R) ≡ σ c1 (σc2 (…(σcn(R))…))
- **Commutativity of σ.** σc1 (σc2(R)) ≡ σc2 (σc1(R))
- **Cascade of π**. In a cascade (sequence) of π operations, all but the last one can be ignored: $\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) \equiv \pi_{List1}(R)$
- **Commuting σ with π**. If the selection condition c involves only those attributes A1, … , An in the projection list, the two operations can be commuted: $\pi_{A1, A2, \ldots, An}(\sigma c(R)) \equiv \sigma c(\pi_{A1, A2, \ldots, An}(R))$
- 5. **Commutativity of (×).** The join operation is commutative, as is the × operation: R×S≡S×R

Use of Selectivities in Cost-Based Optimization

- For this **approach to work**, accurate cost estimates are required so that different strategies can be compared fairly and realistically.
- It uses traditional optimization techniques that search the solution space to a problem for a solution that minimizes an objective (cost) function.
- The cost functions used in query optimization are estimates and not exact cost functions, so the optimization may select a query execution strategy that is not the optimal (absolute best) one.

Semantic Query Optimization

- Uses **constraints** specified on the database schema— such as unique attributes and other more complex constraints—to modify one query into another query that is more efficient to execute.

  e.g SELECT E.Lname, M.Lname
  FROM EMPLOYEE AS E, EMPLOYEE AS M
  WHERE E.Super_ssn=M.Ssn AND E.Salary > M.Salary

✈ If the semantic query optimizer checks for the constraint stated that no employee can earn more than his or her direct supervisor, it does not need to execute the query because it knows that the result of the query will be empty.

Chapter 3

Transaction Processing Concepts

Introduction to Transaction Processing

- **Transaction processing systems** are systems with large databases and hundreds of concurrent users executing database transactions.

  Eg airline reservations, banking

Single-User versus Multiuser Systems

- According to the number of users who can use the system concurrently.
  - **single-user** if at most one user at a time can use the system e.g personal computer sys
  - **Multiuser** if many users can use the system. E.g airline reservations system.

- **Multiprogramming** allows the operating system of the computer to execute multiple programs— or processes—at the same time.
- concurrent execution of processes is actually **interleaved**
  - Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation.
  - Interleaving also prevents a long process from delaying other processes.

✞If the computer system has multiple hardware processors (CPUs), **parallel processing** of multiple processes is possible

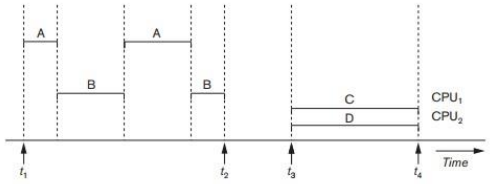


Figure 3.1 Interleaved processing versus parallel processing of concurrent transactions.

Transaction

- A **transaction** is an executing program that forms a logical unit of database processing.
- If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**;
- otherwise it is known as a **read-write transaction.**

Cont'd

- The basic unit of data transfer from disk to main memory is **one disk page** (**disk block**).
- Executing a read_item(X) command includes the following steps:
  1. Find the address of the disk block that contains item X.
  2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer). The size of the buffer is the same as the disk block size.
  3. Copy item X from the buffer to the program variable named X.

- Executing a write_item(X) command includes the following steps:
  1. Find the address of the disk block that contains item X.
  2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
  3. Copy item X from the program variable named X into its correct location in the buffer.
  4. Store the updated disk block from the buffer back to disk (either immediately or at some later point in time).

Why concurrency control?

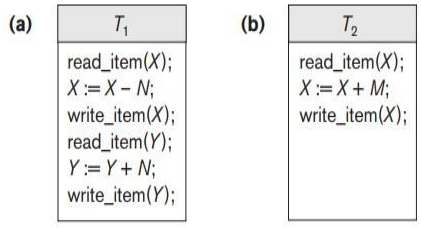


Figure 3.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

E.g for each airline flight each record includes the number of reserved seats on that flight as a named (uniquely identifiable) data item, among other information.
Figure (a) shows a transaction T1 that transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y.
Figure (b) shows a simpler transaction T2 that just reserves M seats on the first flight (X) referenced in transaction T1.

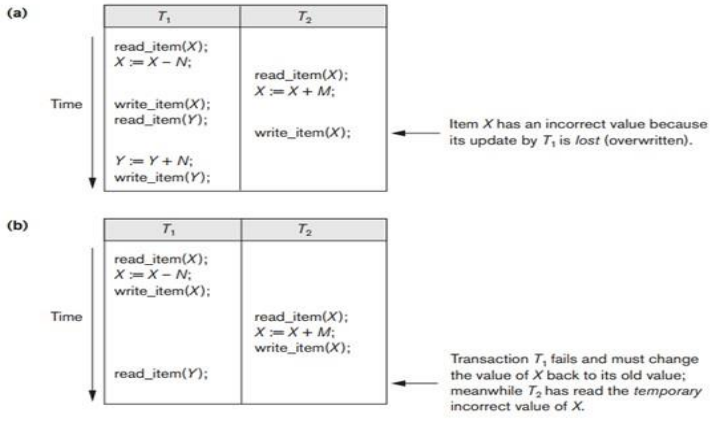
Problems in Concurrency Control

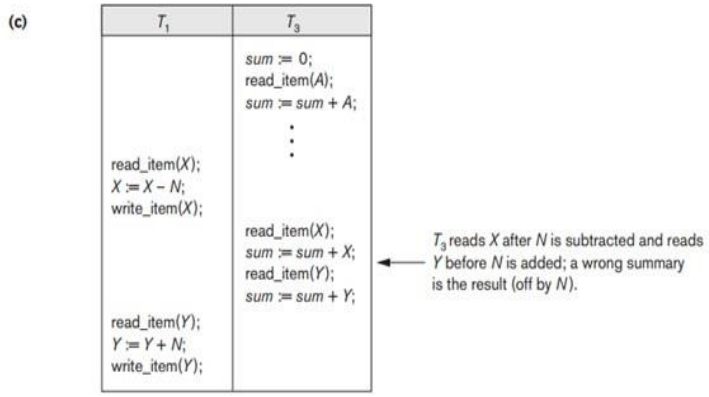
- **The Lost Update Problem**: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.
- **The Temporary Update (or Dirty Read) Problem**: This problem occurs when one transaction updates a database item and then the transaction fails for some reason.

- **The Incorrect Summary Problem:** If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.
- **The Unrepeatable Read Problem:** Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T' between the two reads.

- **The Incorrect Summary Problem:** If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.
- **The Unrepeatable Read Problem:** Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T' between the two reads.

Figure 3.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.





Types of Failures

1. A **computer failure** (system crash): A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.
2. A **transaction** or **system error**: Some operation in the transaction may cause it to fail, such as integer overflow or division by zero.
3. **Local errors** or exception conditions detected by the transaction: During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. E.g data may not found
4. **Concurrency control enforcement** The concurrency control method may abort a transaction because it violates serializability, or it may abort one or more transactions to resolve a state of deadlock among several transactions.
5. **Disk failure:** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6. **Physical problems and catastrophes:** This refers to an endless list of problems that includes power or airconditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Transaction States

- **BEGIN_TRANSACTION:** This marks the beginning of transaction execution.
- **READ** or **WRITE**: These specify read or write operations on the database items that are executed as part of a transaction.
- **END_TRANSACTION**: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.

- **COMMIT_TRANSACTION:** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- **ROLLBACK (or ABORT):** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

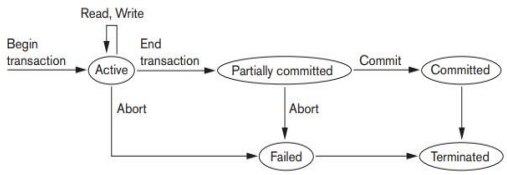


Figure 3.4
State transition diagram illustrating the states for transaction execution.

The System Log

- To be able to recover from failures that affect transactions, the system maintains a log to keep

track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.

- The **log** is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.

log records

1. [start_transaction, T]. Indicates that transaction T has started execution.
2. [write_item, T, X, old_value, new_value]. Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. [read_item, T, X]. Indicates that transaction T has read the value of database item X.
4. [commit, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort, T]. Indicates that transaction T has been aborted.

Commit Point of a Transaction

- Beyond the commit point, the transaction is said to be committed, and its effect must be permanently recorded in the database.

- The transaction then writes a commit record [commit, T] into the log.
- before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called **force-writing the log buffer to disk before commit- ting a transaction.**

ACID Properties of Transactions

- **Atomicity:** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- **Consistency preservation:** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

- **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a

transaction should not be interfered with by any other transactions executing concurrently.

- **Durability or permanency**: 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.

Levels of Isolation:

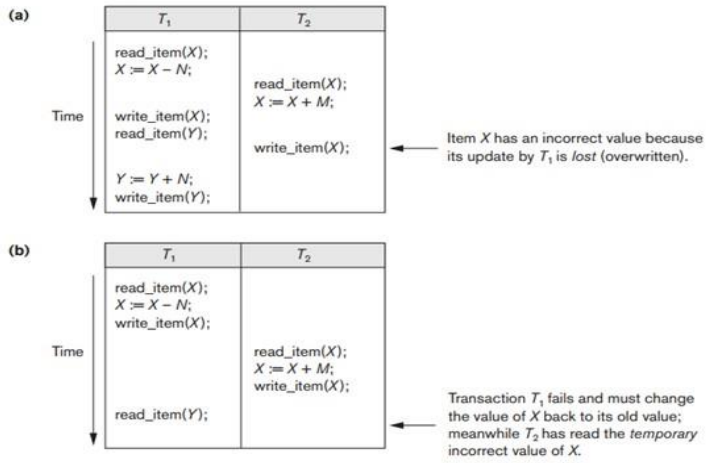
- A transaction is said to have **level 0** (zero) isolation if it does not over- write the dirty reads of higher-level transactions.
- **Level 1** (one) isolation has no lost updates
- **level 2** isolation has no lost updates and no dirty reads.
- **level 3 isolation** (also called true isolation) has, in addition to level 2 properties, repeatable reads.
- Another type of isolation is called **snapshot** isolation, and several practical concurrency control methods are based on this.

Schedules Based on Recoverability

- When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from all the various transactions is known as a **schedule** (or **history**).
- The schedule in Figure 3.3(a), which we shall call Sa, can be written as follows in this notation:

  Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y);

Figure 3.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.



- **Conflicting Operations in a Schedule**: Two operations in a schedule are said to conflict if they satisfy all three of the following conditions:
  (1) They belong to different transactions;
  (2) They access the same item X; and
  (3) At least one of the operations is a write_item(X).
- **read-write conflict** r1(X); w2(X) to w2(X); r1(X)
- **write-write conflict** w1(X); w2(X) to w2(X); w1(X)

Schedules

- once a transaction T is committed, it should never be necessary to roll back T. This ensures that the durability property of transactions is not violated. The schedules that theoretically meet this criterion are called **recoverable schedules**.
- A schedule is said to be **cascadeless**, or to avoid cascading rollback, if every transaction in the schedule reads only items that were written by committed transactions.

- **strict schedule**, in which transactions can neither read nor write an item X until the last transaction that wrote X has committed (or aborted). Strict schedules simplify the recovery process.

Schedules Based on Serializability

- Types of schedules that are always considered to be correct when concurrent transactions are executing are known as **serializable schedules**
- Formally, a schedule S is **serial** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called **nonserial**.

The **problem** with serial schedules

- limit concurrency by prohibiting interleaving of operations.
- In a serial schedule, if a transaction waits for an I/O operation to complete, we cannot switch the CPU processor to another transaction, thus wasting valuable CPU processing time.

- Additionally, if some transaction T is long, the other transactions must wait for T to complete all its operations before starting.

- A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions.
- Two schedules are called **result equivalent** if they produce the same final state of the database.

Equivalence of schedules

- **Conflict Equivalence of Two Schedules:** Two schedules are said to be conflict equivalent if the relative order of any two conflicting operations is the same in both schedules.
- If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on the transactions in the schedule, and hence the schedules are **not conflict equivalent**.

Transaction Support in SQL

- A single SQL statement is always considered to be atomic—either it completes execution without an error or it fails and leaves the database unchanged.
- With SQL, there is no explicit Begin_Transaction statement
- However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK

- The characteristics are the access mode, the diagnostic area size, and the isolation level.
  - The access mode can be specified as READ ONLY or READ WRITE.
  - DIAGNOSTIC SIZE n, specifies an integer value n, which indicates the number of conditions that can be held simultaneously in the diagnostic area.
  - ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE.

- If a transaction executes at a lower isolation level than SERIALIZABLE, then one or more of the following three violations may occur:

1. **Dirty read**. A transaction T1 may read the update of a transaction T2, which has not yet committed.
2. **Nonrepeatable read**. A transaction T1 may read a given value from a table.

Phantoms. A transaction T1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE-clause.

Table 3.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
   READ WRITE
   DIAGNOSTIC SIZE 5
   ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
   VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
   SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

Chapter 4

Concurrency Control Techniques


What is concurrency control?
To ensure serializability of transactions in a multiuser DB

Concurrency Control Problems

Three Main Problems:

-Lost Update

When two transactions update the same data simultaneously and one update is lost

-Uncommitted Data

When two transactions (T1,T2) are executed concurrently, only T2 is recorded

-Inconsistent Retrieval

When one transaction accesses data before and after another transaction completes its operation (The same data)

Purpose of Concurrency Control

- To preserve database consistency
- To resolve read-write and write-write conflicts
- To enforce Isolation(through mutual exclusion)among conflicting transactions

Locking technique

- Some of the main techniques used to control concurrent execution of transactions are based on the concept of locking data items.
- **Locking** is an operation which secures
  - (a) permission to Read
  - (b) permission to Write a data item for a transaction.
- **Types of Locks and System Lock Tables**
  - **I. Binary Locks**
  - **II. Shared/Exclusive (or Read/Write) Locks**

I. Binary Locks

- Are **simple** but are also too **restrictive** for database concurrency control purposes and so are not used much.
- Have two states or values: **locked** and **unlocked** (1 and 0).
- A distinct lock is associated with each database item X.

Locked(1): If the value of the lock on **X is 1**, item X cannot be accessed by a database operation that requests the item.

Unlocked(0): If the value of the lock on **X is 0**, the item can be accessed when requested, and the lock value is changed to 1.

- We refer to the current value (or state) of the lock associated with item X as lock(X).

Two operations, **lock_item** and **unlock_item**, are used with binary locking.

At most one transaction can hold the lock on an item at a given time.

Cont'd

II. Shared/Exclusive (or Read/Write) Locks

- Read operations on the same item are not conflicting.

✦ Allow several transactions to access the same item X if they all access X for **reading purposes only.**

✦ However, if a transaction is to **write** an item X, it must have **exclusive access to X**.

✦ There are **three** locking operations: read_lock(X), write_lock(X), and unlock(X).

Cont'd

✈ A lock associated with an item X, LOCK(X), now has three possible states: **read-locked**, **write-locked**, or **unlocked**.

✈ **A read-locked item** is also called **share-locked** because other transactions are allowed to read the item

✈ **A write-locked item** is called **exclusive-locked** because a single transaction exclusively holds the lock on the item.

Lock conversion

- **Lock upgrade**: change existing read lock to write lock

  if Ti has a read-lock (X) and Tj has no read-lock (X) (i ≠ j) then
  convert read-lock (X) to write-lock (X)
  else
  force Ti to wait until Tj unlocks X

- **Lock downgrade**: change existing write lock to read lock

  If Ti has a write-lock (X) (*no transaction can have any lock on X*)
  convert write-lock (X) to read-lock (X)

Lock compatibility

- **Lock compatibility**
  - A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
  - Any number of transactions can hold shared locks on an item,
    - But if any transaction holds an exclusive lock on the item no other transaction may hold any lock on the item.
  - If a lock cannot be granted, the requesting transaction will be made to wait untill all incompatible locks held by other transactions have been released

- Lock compatibility matrix

	S	X
S	true	false
X	false	false

DBMS lock manager

- The DBMS has **lock manager subsystem** to control locks –

  Lock Manager:
  - Manages locks on data items.
  - Lock table:
    - Lock manager uses it to store the identify of transaction that lock the data item

	D Data item id lo	
T1	X1	Read

Concurrency Control (cont…)

T1	T2	Result
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (X);	T1 followed by T2 write_lock (X);	Write_lock (Y);
X=50, Y=80. read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X); write_item (Y); X=70, Y=50 unlock (X); unlock (Y);		

Discuss (6 minute)

1. The meaning ad conditions which result deadlock ad starvation
2. Identify possible solutions for deadlock and starvation

Deadlock and Starvation

✈ **Deadlock** **:occurs when each transaction T in a set of two or more transactions is waiting for an item that is locked by some other transaction T' in the set**

Example of deadlock situation:

T'1	**T'2**
read_lock (Y);	T'1 and T'2 enter deadlock read_item (Y); read_lock (X); read_item (X);
write_lock (X);	
(waits for X)	write_lock (Y);
	(waits for Y)

Dealing with Deadlock and Starvation

T1 **T2** read_lock (Y); read_item (Y); **unlock (y)**
read_lock (X); read_item (X); **unlock (X)**

write_lock (X); write_lock (Y);

- There is no deadlock in this schedule since T1 unlocks y and T2 unlocks x

Deadlock prevention

- **Deadlock prevention**
  1. **A transaction locks all the needed data items before it begins execution**
     - This way of locking prevents deadlock since a transaction never waits for a data item
     - If any of the items cannot be obtained, none of the items are locked. Rather, the transaction waits and tries again to lock all the items it needs
     - This solution limits concurrency and generally not a practical assumption

Deadlock prevention (cont…)

2. Making a decision about what to do with a transaction involved in a possible deadlock situation:

- **Should it be blocked and made it to wait or should it be aborted**
- **Should the transaction block and abort another transaction**

- These concepts use transaction **timestamp TS(T)** which is a unique identifier assigned to each transaction based on the order they started
- If transaction T1 starts before transaction T2, then TS (T1) < TS(T2)

Deadlock and Starvation (cont…)

- **Deadlock prevention**
  - **Two schemes that prevent dead lock based on time stamp includes wait –die and wound-wait**
  - **Suppose that transaction Ti tries to lock an item X but is not able to do so because X is locked by some other transaction Tj with a conflicting lock. The rules followed by these two schemes are as follows:**
    - **Wait – die: If TS(Ti) < TS(Tj) i.e Ti is older than Tj, then Ti is allowed to wait ; other wise, abort Ti (Ti dies) and restart it later with the same time stamp**
    - **Wound - wait: If TS(Ti) < TS(Tj), abort Tj (Ti wounds Tj) and restart it later with the same timestamp; other wise Ti is allowed to wait.**

Starvation

- Another problem that may occur when we use locking is **starvation**,.
- which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
- This may occur if the waiting scheme for locked items is unfair in that it gives priority to some transactions over others.
- **One solution for starvation** is to have a fair waiting scheme, such as using a **first-come-first-served queue**; transactions are enabled to lock an item in the order in which they originally requested the lock.

2. Concurrency control based on Timestamp ordering

- **Timestamp (TS)**
  - **Time stamp** is a unique identifier created by the DBMS to identify a transaction
  - A larger timestamp value indicates a more recent event or operation
  - **Time stamp ordering algorithm associates two time stamp values (TS) with each database item X**
    1. Read_TS(x) : the read time stamp of x: This is the largest timestamp among all the timestamps of transactions that have successfully read item x
  - Read_TS(X)=TS(T) where T is the youngest transaction that has read X successfully

2. Write_TS(X) : This the largest of all the timestamps of transactions that have successfully written item x – that is, write_TS(x) = TS(T), where T is the youngest transaction that has written x successfully

Timestamp ordering (cont…)

I. **Basic Timestamp Ordering (TO):**
   – whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated.

II. **Strict Timestamp Ordering (TO).** A variation of basic TO called strict TO ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable.
   – This algorithm does not cause deadlock.

3. Multiversion concurrency control techniques

- This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction.
- Unlike other mechanisms a read operation in this mechanism is never rejected – **Side effect:**
  - Need more storage (RAM and disk) is required to maintain multiple versions
  - To check unlimited growth of versions, a garbage collection is run when some criteria is satisfied

Multiversion techniques (cont…)

- Assume X1, X2, …, Xn are the version of a data item X created by a write operation of transactions.
- With each Xi a read_TS (read timestamp) and a write_TS (write timestamp) are associated.

– **read_TS(Xi)**: The read timestamp of Xi is the largest of all the timestamps of transactions that have successfully read version Xi.

– **write_TS(Xi)**: The write timestamp of Xi that wrote the value of version Xi.

4. Validation (Optimistic) Concurrency Control Schemes

- ✈ In this technique, serializability is checked only at the time of commit and transactions are aborted in case of nonserializable schedules
- ✈ No checking is done while transaction is executing
- ✈ In this scheme, updates in the transaction are not applied directly to the database item until it reaches its commit point

• There are three phases:

1. **Read phase**
2. **Validation phase**
3. **Write phase**

Cont'd

1. Read phase:

– A transaction can read values of committed data items. However, updates are applied only to local copies (versions) of the data items (in database cache)

2. Validation phase: Serializability is checked before transactions write their updates to the database.

3. Write phase: On a successful validation transactions' updates are applied to the database; otherwise, transactions are restarted

5. Granularity of data items and Multiple Granularity Locking (MGL)

- A lockable unit of data defines its granularity.
- **Granularity** can be coarse (entire database) or it can be fine (a tuple or an attribute of a relation).
- Data item granularity significantly affects concurrency control performance.
- Thus, the degree of concurrency is low for coarse granularity and high for fine granularity.

- Example of data item granularity:
  1. A field of a database record (an attribute of a tuple)
  2. A database record (a tuple or a relation)
  3. A database table
  4. The entire database

Granularity of data items and Multiple Granularity Locking (Cont…)

- Lock granularity affects concurrency in the following manner:
  - The larger the lock granularity used, the more concurrency is reduced.
  - This means that row-level locking maximizes concurrency because it leaves all but one row on the page unlocked.
  - On the other hand, system overhead is increased because each locked row requires one lock.
  - Page level locking (and table-level locking) restricts the availability of data but decreases the system overhead.

Chapter 5

Database Recovery Techniques

Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve consistency and durability property of transactions.

128

Types of Failure

The database may become unavailable for use due to

- **Transaction failure:** Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- **System failure:** System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- **Media failure:** Disk head crash, power disruption, etc.

129

Transaction Log

- For recovery from any type of failure, data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequential file called **Transaction log**.
  - Sample log entry

T ID	BOperation	Data item	BFIM	AFIM
T1	Begin			
T1	Write	X	X = 100	X = 200
T2	Begin			
T1	W	Y	Y = 50	Y = 100
T1	R	M	M = 200	M = 200
T3	R	N	N = 400	N = 400
T1	End			

130

Caching of disk blocks

•The flushing is controlled by **dirty** and **PinUnpin** bits associated with each buffer in the cache

- ✦The dirty bits are used to indicate whether or not the buffer has been modified (1=modified, 0=not modified)
- ✦As soon as the buffer is modified, the dirty bits for the corresponding cache directory entry are set to 1
- ✦A page in the cache is **pinned** (bit value 1) if it can not be written back to disk

131

Cont'd

- **Two main strategies can be employed when flushing a modified buffer back to disk**
- **In-place update**:
- Writes the buffer back to the same original disk location
- The disk version (old value) of any changed data item is overwritten by the cache version (new value).
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

132

Write-Ahead Logging

When **in-place** update is used, then log is necessary for recovery and it must be available to recovery manager.

- This is achieved by **Write-Ahead Logging** (WAL) protocol.
- WAL states that :
  - **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM), its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  - **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

133

Steal/No-Steal and Force/No-Force

Standard DBMS recovery includes these terminologies for specifying when a page from the DB can be written to disk from the cache

– Possible ways for flushing database cache to database disk:

1. **Steal:** Cache can be flushed before transaction commits.
   - ✓When a DBMS buffer manager needs a buffer frame for another transaction
2. **No-Steal:** Cache cannot be flushed before transaction commit.
3. **Force:** Cache is immediately flushed (forced) to disk when the transaction commits .
4. **No-Force:** Cache is deferred until transaction commits

134

Cont'd

– These give rise to four different ways for handling recovery:

- Steal/No-Force
- Steal/Force
- No-Steal/No-Force
- No-Steal/Force

Checkpointing

✦Another type of entry in the log is a checkpoint

✦A check point record is written into the log periodically at the point when the system flush all DBMS buffers that haven been modified

✦Used to minimize the task of recovery

✦The recovery manager of a DBMS must decide at what interval to take a checkpoint

Cont'd

- The following steps defines a checkpoint operation:
  1. Suspend execution of transactions temporarily.
  2. Enforce writing of modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.

  During recovery, **redo** or **undo** is required to transactions appearing after [checkpoint] record.

Transaction Roll-back (Undo)

- When a transaction fails for what ever reason after updating the database, it may be necessary to rollback the transaction
- If any data item values have been changed by the transaction and written to the database, they must be restored to their previous values
- The undo type log entries are used to restore the old values of data items that must be rolled back

Recovery Scheme

Deferred Update (No Undo/Redo)

- We can state a typical deferred update protocol as follows:
  - A transaction can not change the database on disk until it reaches its commit point
  - A transaction does not reach its commit point until all of its update operations are recorded into the log and the log is written to disk

139

Database Recovery (cont…)

Recovery using Deferred Update

- The data update goes as follows:

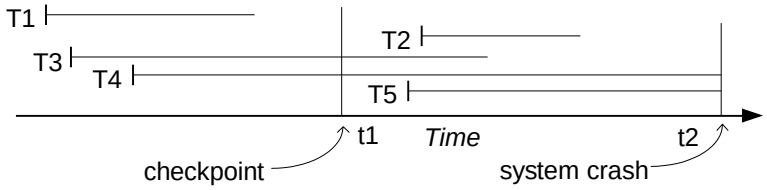
1. A set of transactions record their updates in the log.
2. At commit point under WAL scheme, these updates are saved on database disk.

- After reboot from a failure the log is used to redo all the transactions affected by this failure.
- No undo is required because no AFIM is flushed to the disk before a transaction commits.

140

Database Recovery (cont…)

- *Multiuser environment* requires some concurrency control mechanism to guarantee isolation property of transactions.
- In a system recovery, transactions which were recorded in the log after the last checkpoint were redone.
- The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.



Recovery in a concurrent users environment.

141

Database Recovery (cont…)

Recovery Techniques Based on Immediate Update

Undo/No-redo Algorithm

- In this algorithm, AFIMs of a transaction are flushed to the database disk before it commits.
- For this reason, the recovery manager undoes all transactions during recovery.
- No transaction is redone.

142

Database Recovery (cont…)

Recovery Techniques Based on Immediate Update

Undo/Redo Algorithm

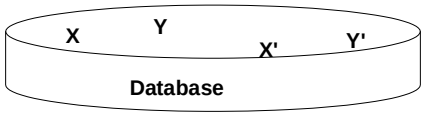
- Recovery schemes of this category apply undo and also redo for recovery.
- Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
- Commit table records transactions to be committed and active table records active transactions.
- To minimize the work of the recovery manager checkpointing is used
- The recovery manager performs:
  - Undo of a transaction if it is in the active table
  - Redo of a transaction if it is in the commit table.

143

Database Recovery (cont…)

Shadow Paging

- The AFIM does not overwrite its BFIM but recorded at another place on the disk.
- Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.
- This recovery scheme does not require the use of a log in a single user environment
- Multiuser environment may need the log for concurrency control



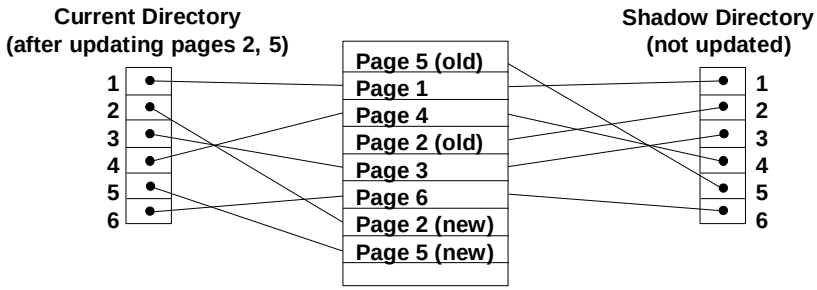
X and Y: Shadow copies of data items (old)
X` and Y`: Current copies of data items (new)

144

Database Recovery (cont…)

Shadow Paging …

- To manage access of data items by concurrent transactions, two directories (current and shadow) are used.
- The directory arrangement is illustrated below. Here a page is a data item.
- Example :



145

Chapter 6

Database Security and Authorization

6.1 Introduction to DB Security Issues

- Security issues
  - **Legal and ethical issues**: The right to access certain information.
  - **Policy issues**: At the governmental, institutional and corporate level.
  - **System-related issues:** Such as the system level at which various security functions should be enforced. For example, Hardware, OS or DBMS level.

147

Threats to databases

- **Loss of integrity:**
  - Database integrity refers to the requirement that information be protected from improper **modification**.
  - Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts.
- **Loss of availability**
  - Database availability refers to making objects **available** to a human user or a program who/which has a legitimate right to those data objects.
  - Loss of availability occurs when the user or program cannot access these objects.
- **Loss of confidentiality**
  - Database confidentiality refers to the protection of data from **unauthorized disclosure**.

148

Counter measurements

- To protect databases against these types of threats, four kinds of countermeasures can be implemented:
  - **Access control**
  - **Inference control**
  - **Flow control – Encryption**

149

Two types of database security mechanisms

- **Discretionary security mechanisms**. These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- **Mandatory security mechanisms.** These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization.
- For example, a typical security policy is to permit users at a certain classification (or clearance) level to see only the data items classified at the user's own (or lower) classification level. An extension of this is role-based security, which enforces policies and privileges based on the concept of orga- nizational roles.

150

Control Measures

- Four main control measures are used to provide security of data in databases:
  - Access control -- Inference control
  - Flow control -- Data encryption

I. **Access control,** is the security mechanism of a DBMS must include provisions for restricting access to the database as a whole.
   - **AC** handled by creating user accounts and passwords to control login process by the DBMS.

II. **Inference control measures,** handles Statistical database security problem

For example, a database for population statistics may provide statistics based on age groups, income levels, household size, education levels, and other criteria.

151

Cont'd

III. **Flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
   - **Covert channels**, Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization

IV. **Data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type of communication network

152

The DBA has a DBA account in the DBMS

- Sometimes these are called a system or superuser account
- These accounts provide powerful capabilities such as:
  1. Account creation
  2. Privilege granting
  3. Privilege revocation
  4. Security level assignment
- Action 1 is **access control**, whereas 2 and 3 are **discretionary** and 4 is used to **control mandatory** authorization

153

Access Protection, User Accounts, and Database Audits

- Whenever a person or group of persons need to access a database system, the individual or group must first apply for a user account
  - The DBA will then create a new **account id** and **password** for the user if he/she deems there is a legitimate need to access the database
- The user must log in to the DBMS by entering account id and password whenever database access is needed

154

Cont'd

- The database system must also keep **track of all operations** on the database that are applied by a certain user throughout **each login session**
  - To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify **system log**, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash

155

Cont'd

- If any tampering with the database is suspected, a **database audit** is performed
  - A database audit consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.

156

6.2. Discretionary Access Control

- This is the typical method of enforcing **access control** in a database based on the **granting** and **revoking** of **privileges**.

Types of Discretionary Privileges

- The **account level**:
- At this level, the DBA **specifies** the particular privileges that each account holds independently of the relations in the database.
- The **relation level** (or **table level**):
- At this level, the DBA can **control** the privilege to access each individual relation or view in the database.

157

Discretionary Privileges (cont…)

- The Privileges at the **account level** apply to the capabilities provided to the account itself and can include
  - the **CREATE SCHEMA** or **CREATE TABLE** privilege, to create a schema or base relation;
  - the **CREATE VIEW** privilege;
  - the **ALTER** privilege, to apply schema changes such as adding or removing attributes from relations;
  - the **DROP** privilege, to delete relations or views;
  - the **MODIFY** privilege, to insert, delete, or update tuples;
  - and the **SELECT** privilege, to retrieve information from the database by using a **SELECT** query.

158

Cont'd

- The second level of privileges applies to the **relation level**
  - This includes privileges on **base relations** and virtual (**view**) relations.
- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the **access matrix model** where
  - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
  - The **columns** represent **objects** (relations, records, columns, views, operations).
  - Each position **M(i,j)** in the matrix represents the types of privileges (read, write, update) that **subject i** holds on **object j**.

159

Cont'd

- To control the granting and revoking of relation privileges, each relation R in a database is assigned an **owner account**, which is typically the account that was used when the relation was created in the first place.
  - The owner of a relation is given all privileges on that relation.
  - DBA can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the **CREATE SCHEMA** command.
  - The owner account holder can **pass privileges** on any of the owned relation to other users by **granting** privileges to their accounts.

160

Cont'd

- In SQL, the following types of privileges can be granted on each individual relation R:
  - **SELECT** (retrieval or read) privilege on R:
    - Gives a retrieval privilege to the account
    - This gives the account holder the privilege to use the **SELECT** statement to retrieve tuples from R.
  - **MODIFY** privileges on R:
    - This gives the account the capability to modify tuples of R.
    - This privilege is further divided into **UPDATE**, **DELETE**, and **INSERT** privileges to apply the corresponding SQL command to R.
    - In addition, both the **INSERT** and **UPDATE** privileges can specify that only certain attributes can be updated by the account

161

Cont'd

- The following types of privileges can be granted on each individual relation R :
  - **REFERENCES** privilege on R:
    - This gives the account holder the right to **reference** relation R when specifying integrity constraints
    - The privilege can also be **restricted** to specific attributes of R

Notice that to create a **view**, the account must have **SELECT** privilege on all relations involved in the view definition

162

Specifying Privileges Using Views

- The mechanism of **views** is an important discretionary authorization mechanism. For example,
  - If the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that includes only those attributes and then grant SELECT on V to B
  - The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access

163

Revoking Privileges

- In some cases, it is desirable to grant a privilege to a user temporarily. For example,
  - The owner of a relation may want to grant the **SELECT** privilege to a user for a specific task and then revoke that privilege once the task is completed
  - Hence, a mechanism for **revoking** privileges is needed.
  - In SQL, a **REVOKE** command is included for the purpose of **canceling privileges**.

164

Propagation of Privileges using the GRANT OPTION

- Whenever the owner A of a relation R grants a privilege on R to another account B, privilege can be given to B with or without the **GRANT OPTION**.
- If the **GRANT OPTION** is given, this means that B can also grant that privilege on R to other accounts.
  - Suppose that B is given the **GRANT OPTION** by A and that B then grants the privilege on R to a third account C, also with **GRANT OPTION**.
  - In this way, privileges on R can **propagate** to other accounts without the knowledge of the owner of R.
  - If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege should automatically be revoked by the system.

165

Example

- Suppose that the DBA creates four accounts
  - A1, A2, A3, A4
- and wants only A1 to be able to create base relations. Then, the DBA must issue the following GRANT command in SQL

```
GRANT CREATE TABLE TO A1;
```

- The same effect can be accomplished by having the DBA issue a **CREATE SCHEMA** command as follows:

```
CREATE SCHAMA schema1 AUTHORIZATION A1;
```

166

An Example (cont…)

- User account A1 can create tables under the schema called **shema1**.
- Suppose that A1 **creates** the two base relations **EMPLOYEE** and **DEPARTMENT**
  - A1 is then **owner** of these two relations and hence all the relation privileges on each of them.
- Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations, but A1 does not want A2 to be able to propagate these privileges to additional accounts:

```
GRANT INSERT, DELETE ON
    EMPLOYEE, DEPARTMENT TO A2;
```

167

An Example (cont…)

- Suppose that A1 wants to allow A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts.
- A1 can issue the command:

```
GRANT SELECT ON EMPLOYEE, DEPARTMENT
    TO A3 WITH GRANT OPTION;
```

- A3 can grant the **SELECT** privilege on the **EMPLOYEE** relation to A4 by issuing:

```
GRANT SELECT ON EMPLOYEE TO A4;
```

  – Notice that A4 can't propagate the SELECT privilege because GRANT OPTION was not given to A4

168

An Example (cont…)

- Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 can issue:

```
REVOKE SELECT ON EMPLOYEE FROM A3;
```

- The DBMS must now automatically revoke the SELECT privilege on EMPLOYEE from A4, too, because A3 granted that privilege to A4 and A3 does not have the privilege any more.

169

Example (cont…)

- Suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege.
  - The limitation is to retrieve only the NAME, BDATE, and ADDRESS attributes and only for the tuples with DNO=5.
- A1 then create the view:

```
CREATE VIEW A3EMPLOYEE AS
  SELECT NAME, BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE DNO = 5;
```

- After the view is created, A1 can grant **SELECT** on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3
   WITH GRANT OPTION;
```

170

An Example (cont…)

- Finally, suppose that A1 wants to allow A4 to update only the SALARY attribute of EMPLOYEE;
- A1 can issue:

```
GRANT UPDATE ON EMPLOYEE (SALARY) TO
A4;
```

  - The **UPDATE** or **INSERT** privilege can specify particular attributes that may be updated or inserted in a relation

171

6.3. Mandatory Access Control and Role-Based Access Control for Multilevel Security

- The discretionary access control techniques of granting and revoking privileges on relations has been the main security mechanism for relational database systems.
- This is an all-or-nothing method: – A user either has or does not have a certain privilege.
- In many applications, **additional security policy** is needed that classifies data and users based on security classes.
  - This approach as **mandatory access control**, would typically be **combined** with the discretionary access control mechanisms.

172

Cont'd

- Typical **security classes** are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest: TS ≥ S ≥ C ≥ U
- The commonly used model for multilevel security, classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications, T, S, C, or U:
  - **Clearance** (classification) of a subject S as **class(S)** and to the **classification** of an object O as **class(O)**.
- Two restrictions are enforced on data access based on the subject/object classifications:
  - **Simple security property:** A subject S is not allowed read access to an object O unless class(S) ≥ class(O)

173

Mandatory Access Control and Role-Based Access Control for

Multilevel Security (cont…)

- To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects.
- Hence, each attribute A is associated with a **classification attribute C** in the schema, and each attribute value in a tuple is associated with a corresponding security classification.
- In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole.

174

Comparing Discretionary Access Control and Mandatory Access Control

- **Discretionary Access Control (DAC)** policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains.
  - The main drawback of **DAC** models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs

175

Comparing Discretionary Access Control and

Mandatory Access Control (cont…)

- By contrast, mandatory policies ensure a high degree of protection in a way, they prevent any illegal flow of information.
- Mandatory policies have the drawback of being too rigid and they are only applicable in limited environments.
- In many practical situations, discretionary policies are preferred because they offer a better tradeoff between security and applicability.

176

Role-Based Access Control

- **Role-based access control (RBAC)** has emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise wide systems.
- Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles.
- Roles can be created using the **CREATE ROLE** commands.
  - The **GRANT** and **REVOKE** commands discussed under DAC can then be used to assign and revoke privileges from roles

177

Role-Based Access Control (cont…)

- **RBAC** appears to be a viable alternative to discretionary and mandatory access controls; it ensures that only authorized users are given access to certain data or resources.
- Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.
- Role hierarchy in **RBAC** is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.

178

Role-Based Access Control (cont…)

- Using **RBAC** model is highly desirable goal for addressing the key security requirements of Web-based applications.
- In contrast, discretionary access control (**DAC**) and mandatory access control (**MAC**) models **lack capabilities** needed to support the security requirements emerging enterprises and Web-based applications.

179

Introduction to Statistical
Database Security

- **Statistical databases** are used mainly to produce statistics on various populations.
- The database may contain **confidential data** on individuals, which should be protected from user access.
- Users are permitted to retrieve **statistical information** on the populations, such as **averages, sums, counts, maximums, minimums,** and **standard deviations**.
- A **population** is a set of tuples of a relation (table) that satisfy some selection condition.
- Statistical queries involve applying **statistical functions** to a **population** of tuples.

180

Statistical database Security (cont…)

- For example, we may want to retrieve the *number* of individuals in a **population** or the *average income* in the population.
  - However, statistical users are not allowed to retrieve individual data, such as the income of a specific person.
- Statistical database security techniques must prohibit the retrieval of individual data.
- This can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION.
  - Such queries are sometimes called **statistical queries**.

181

Statistical database Security (cont…)

- It is DBMS's responsibility to ensure confidentiality of information about individuals, while still providing useful statistical summaries of data about those individuals to users.
- Provision of **privacy protection** of users in a statistical database is paramount.
- In some cases it is possible to **infer** the values of individual tuples from a sequence statistical queries.
  - This is particularly true when the conditions result in a population consisting of a small number of tuples.

182

Questions

- Difference between parallel processing and interleaving (1 pt)
- Write Advantage of interleaving(2 pt)
- Why concurrency control? (2 pt)
- List and describe ACID Properties of Transactions(2 pt)
- When operations in a schedule are said to conflict? (2 pt)
- When schedule is said to be **cascadeless? (1 pt)**