# Data Persistence

CS 340

# Persistence Strategies

- There are many strategies a program can use for persisting its in-memory object model
  - Approach #1 – Full in-memory object model with bulk updates
  - Approach #2 – Full in-memory object model with incremental updates
  - Approach #3 – Partial in-memory object model with incremental updates

# Full in-memory object model with bulk updates

- Load full object model from disk into memory
- Application features operate on in-memory object model
- Save full object model to disk at appropriate times ("Save", application exit, etc.)
- Crash causes data loss
- Full in-memory model and bulk load/save is not feasible for large data sets

# Full in-memory object model with incremental updates

- Load full object model from disk into memory
- Application features operate on in-memory object model
- Incremental changes to the in-memory object model are immediately saved to disk
- Full in-memory model and bulk load is not feasible for large data sets

# Partial in-memory model with incremental updates

- Full object model exists only on disk (not in memory)
- Application dynamically loads a subset of the object model from disk as needed to perform an operation.
- Incremental changes to the partial in-memory object model are immediately saved to disk
- The partial in-memory object model is discarded when the operation is complete
- Scales to large data sets
- Takes work to fetch the data required for each operation

# Persistence Technologies

- Persistence Options
  - Serialization
  - XML
  - Custom file format
  - Database
  - Cloud storage services (Amazon, Microsoft, Google, …)
- Each of these approaches is appropriate in different contexts
- Database advantages
  - Easy to use
  - Allows incremental updates
  - Allows concurrent data sharing by multiple users and programs
- Relational Databases are the most common

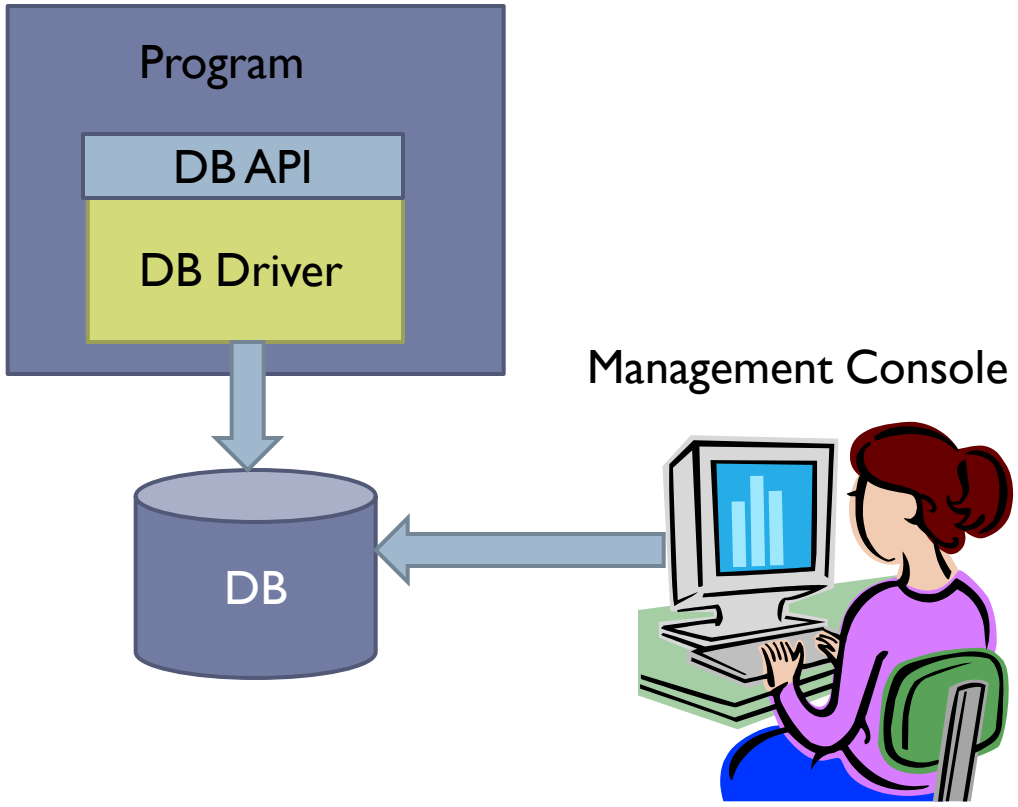
# Database Management Systems (DBMS)

- Databases are implemented by software systems called Database Management Systems (DBMS)
- Commonly used Relational DBMS's include MySQL, MS SQL Server, and Oracle
- DBMS's store data in files in a way that scales to large amounts of data and allows data to be accessed efficiently

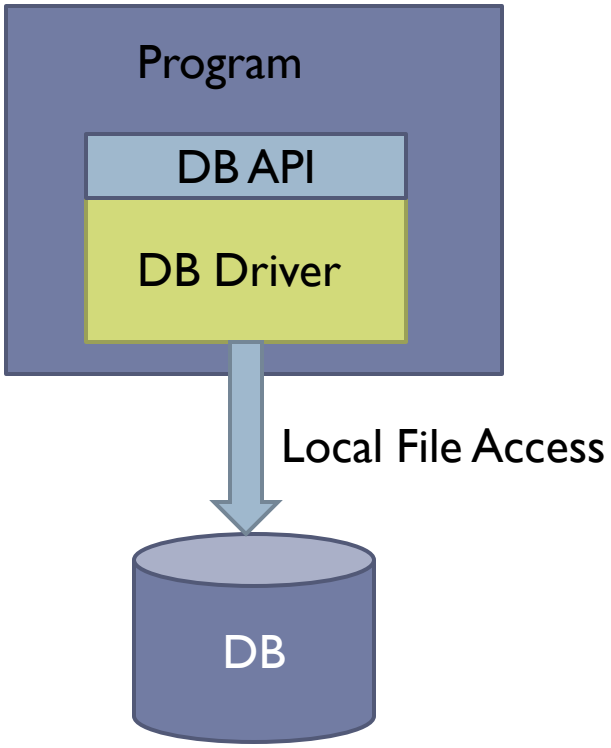
# Programmatic vs. Interactive Database Access

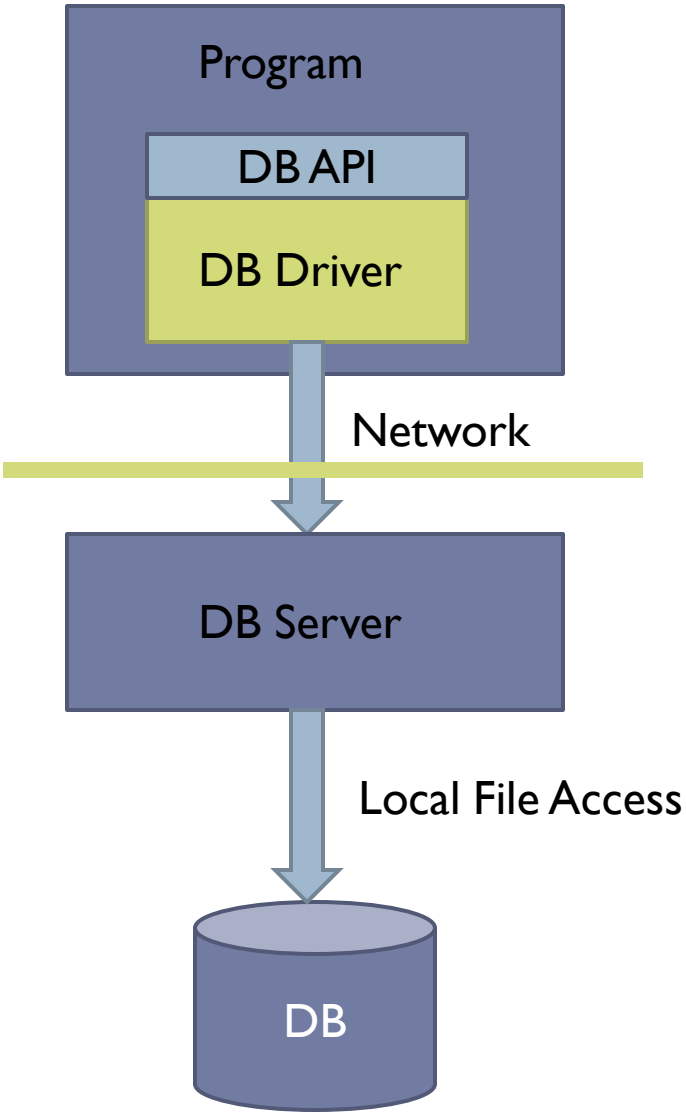
Programs can access a database through APIs such as ADO.NET or JDBC.

End users can access a database through an interactive management application that allows them to query and modify the database.



# Embedded vs. Client/Server




Some DBMS's are Embedded only.
Some are Client/Server only.
Some can work in either mode.

# Relational Databases

- Relational databases use the relational data model you learned about in CS 236
- In the object-oriented data model we have classes. Objects are instances of classes. Objects have attributes. Relationships between objects are represented as pointers.
- In the relational data model, data is stored in tables consisting of columns and rows. Each row in a table represents an object. The columns in a row store the object's attributes.
- Each object has a "key", which is a unique identifier for that object. Relationships between objects are represented using keys.
- Taken together, all the table definitions in a database make up the "schema" for the database.

# Book Club Schema

member

| id | name | email_address |
|---|---|---|
| 1 | 'Ann' | 'ann@cs.byu.edu' |
| 2 | 'Bob' | 'bob@cs.byu.edu' |
| 3 | 'Chris' | 'chris@cs.byu.edu' |

book

| id | title | author | genre |
|---|---|---|---|
| 1 | 'Decision Points' | 'George W. Bush' | 'NonFiction' |
| 2 | 'The Work and the Glory' | 'Gerald Lund' | 'HistoricalFiction' |
| 3 | 'Dracula' | 'Bram Stoker' | 'Fiction' |
| 4 | 'The Holy Bible' | 'The Lord' | 'NonFiction' |

reading

| member_id | book_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 2 |
| 2 | 3 |
| 3 | 3 |
| 3 | 4 |

# Book Club Schema

category

| id | name | parent_id |
|---|---|---|
| 1 | 'Top' | Null |
| 2 | 'Must Read' | 1 |
| 3 | 'Must Read (New)' | 2 |
| 4 | 'Must Read (Old)' | 2 |
| 5 | 'Must Read (Really Old)' | 2 |
| 6 | 'Optional' | 1 |
| 7 | 'Optional (New)' | 6 |
| 8 | 'Optional (Old)' | 6 |
| 9 | 'Optional (Really Old)' | 6 |

category_book

| category_id | book_id |
|---|---|
| 7 | 1 |
| 3 | 2 |
| 8 | 3 |
| 5 | 4 |

book_club

| next_member_id | next_book_id | next_category_id |
|---|---|---|
| 4 | 5 | 10 |

# SQL – Structured Query Language

- Language for performing relational database operations
  - Create tables
  - Delete tables
  - Insert rows
  - Update rows
  - Delete rows
  - Query for matching rows
  - Much more …

# SQL Data Types

- BIGINT
- BLOB
- CHAR
- CHAR FOR BIT DATA
- CLOB
- DATE
- DECIMAL
- DOUBLE
- DOUBLE PRECISION
- FLOAT
- INTEGER
- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- NUMERIC
- REAL
- SMALLINT
- TIME
- TIMESTAMP
- VARCHAR
- VARCHAR FOR BIT DATA

# Creating and Deleting Tables

- CREATE TABLE
  - Book Club Example
  - NULL
  - Primary Keys
- DROP TABLE
  - Book Club Example

# Modeling Object Relationships

- Connections between objects are represented using foreign keys
- Foreign Key: A column in table $T_1$ stores primary keys of objects in table $T_2$
- Book Club Examples
  - Reading table stores Member and Book keys
  - Category table stores parent Category key
  - Category_Book table stores Category and Book keys

# Modeling Object Relationships

- Types of Object Relationships
  - One-to-One
    - A Person has one Head; A Head belongs to one Person
    - Either table contains a foreign key referencing the other table
  - One-to-Many
    - A Category has many sub Categories; a Category has one parent Category
    - The "Many" table contains a foreign key referencing the "One" table
  - Many-to-Many
    - A Member has read many Books; A Book has been read by many Members
    - A Category contains many Books; A Book belongs to many Categories
    - Create a "junction table" whose rows contain foreign keys of related objects

# Inserting Data into Tables

- INSERT
  - Book Club Example

# Queries

```
SELECT  Column, Column, …
FROM  Table, Table, …
WHERE  Condition
```

# Queries

book

| id | title | author | genre |
|---|---|---|---|
| 1 | 'Decision Points' | 'George W. Bush' | 'NonFiction' |
| 2 | 'The Work and the Glory' | 'Gerald Lund' | 'HistoricalFiction' |
| 3 | 'Dracula' | 'Bram Stoker' | 'Fiction' |
| 4 | 'The Holy Bible' | 'The Lord' | 'NonFiction' |

List all books

```
SELECT *
FROM book
```

result

| id | title | author | genre |
|---|---|---|---|
| 1 | 'Decision Points' | 'George W. Bush' | 'NonFiction' |
| 2 | 'The Work and the Glory' | 'Gerald Lund' | 'HistoricalFiction' |
| 3 | 'Dracula' | 'Bram Stoker' | 'Fiction' |
| 4 | 'The Holy Bible' | 'The Lord' | 'NonFiction' |

# Queries

book

| id | title | author | genre |
|---|---|---|---|
| 1 | 'Decision Points' | 'George W. Bush' | 'NonFiction' |
| 2 | 'The Work and the Glory' | 'Gerald Lund' | 'HistoricalFiction' |
| 3 | 'Dracula' | 'Bram Stoker' | 'Fiction' |
| 4 | 'The Holy Bible' | 'The Lord' | 'NonFiction' |

List the authors and titles of all non-fiction books

```
SELECT  author, title
FROM  book
WHERE  genre = 'NonFiction'
```

result

| author | title |
|---|---|
| 'George W. Bush' | 'Decision Points' |
| 'The Lord' | 'The Holy Bible' |

# Queries

category

| id | name | parent_id |
|---|---|---|
| 1 | 'Top' | Null |
| 2 | 'Must Read' | 1 |
| 3 | 'Must Read (New)' | 2 |
| 4 | 'Must Read (Old)' | 2 |
| 5 | 'Must Read (Really Old)' | 2 |
| 6 | 'Optional' | 1 |
| 7 | 'Optional (New)' | 6 |
| 8 | 'Optional (Old)' | 6 |
| 9 | 'Optional (Really Old)' | 6 |

List the sub-categories of category 'Top'

```
SELECT  id, name, parent_id
FROM  category
WHERE  parent_id = 1
```

result

| id | name | parent_id |
|---|---|---|
| 2 | 'Must Read' | 1 |
| 6 | 'Optional' | 1 |

# Queries

List the books read by each member

```
SELECT  member.name,  book.title
FROM  member,  reading,  book
WHERE  member.id = reading.member_id AND
       book.id = reading.book_id
```

JOIN

member X reading X book (3 x 6 x 4 = 72 rows)

| member.id | member.name | member.email_address | reading.member_id | reading.book_id | book.id | book.title | book.author | book.genre |
|---|---|---|---|---|---|---|---|---|
| 1 | 'Ann' | 'ann@cs.byu.edu' | 1 | 1 | 1 | 'Decision Points' | 'George W. Bush' | 'NonFiction' |
| 1 | 'Ann' | 'ann@cs.byu.edu' | 1 | 1 | 2 | 'The Work and the Glory' | 'Gerald Lund' | 'HistoricalFiction' |
| 1 | 'Ann' | 'ann@cs.byu.edu' | 1 | 1 | 3 | 'Dracula' | 'Bram Stoker' | 'Fiction' |
| 1 | 'Ann' | 'ann@cs.byu.edu' | 1 | 1 | 4 | 'The Holy Bible' | 'The Lord' | 'NonFiction' |
| … | … | … | … | … | … | … | … | … |

# Queries

List the books read by each member

```
SELECT  member.name,  book.title
FROM  member,  reading,  book
WHERE  member.id = reading.member_id AND
            book.id = reading.book_id
```

result

| name | title |
|---|---|
| 'Ann' | 'Decision Points' |
| 'Ann' | 'The Work and the Glory' |
| 'Bob' | 'The Work and the Glory' |
| 'Bob' | 'Dracula' |
| 'Chris' | 'Dracula' |
| 'Chris' | 'The Holy Bible' |

# Updates

```
UPDATE  Table
SET  Column = Value, Column = Value, …
WHERE  Condition
```

Change a member's information

```
UPDATE  member
SET name = 'Chris Jones',
        email_address = 'chris@gmail.com'
WHERE  id = 3
```

Set all member email addresses to empty

```
UPDATE  member
SET email_address = ''
```

# Deletes

```
DELETE FROM Table
WHERE  Condition
```

Delete a member

```
DELETE FROM member
WHERE  id = 3
```

Delete all readings for a member

```
DELETE FROM reading
WHERE  member_id = 3
```

Delete all books

```
DELETE FROM book
```

# Database Transactions

- Database transactions have the ACID properties
  - A = Atomic
    - Transactions are "all or nothing". Either all of the operations in a transaction are performed, or none of them are. No partial execution.
  - C = Consistent
    - When multiple transactions execute concurrently, the database is kept in a consistent state.
    - Concurrent transactions $T_1$ and $T_2$ are "serialized". The final effect will be either $T_1$ followed by $T_2$ or $T_2$ followed by $T_1$.
  - I = Isolated
    - Concurrent transactions are isolated from each other. Changes made by a transaction are not visible to other transactions until the transaction commits.
  - D = Durable
    - The changes made by a committed transaction are permanent.

# Programmatic Database Access

- Open a database connection
- Start a transaction
- Execute queries and/or updates
- Commit or Rollback the transaction
- Close the database connection

# Open a Database Connection / Start a Transaction

```
import java.sql.*;

String dbName = "db" + File.separator + "bookclub";
String connectionURL = "jdbc:derby:" + dbName + ";create=false";

Connection connection = null;
try {
    // Open a database connection
    connection = DriverManager.getConnection(connectionURL);

    // Start a transaction
    connection.setAutoCommit(false);
}
catch (SQLException e) {
    // ERROR
}
```

# Execute a Query

```
PreparedStatement stmt = null;
ResultSet rs = null;
try {
    String sql = "select id, title, author, genre from book";
    stmt = connection.prepareStatement(sql);

    rs = stmt.executeQuery();
    while (rs.next()) {
        int id = rs.getInt(1);
        String title = rs.getString(2);
        String author = rs.getString(3);
        Genre genre = convertGenre(rs.getString(4));
    }
}
catch (SQLException e) {
    // ERROR
}
finally {
    if (rs != null) rs.close();
    if (stmt != null) stmt.close();
}
```

# Execute an Update

```
PreparedStatement stmt = null;
try {
    String sql = "update book " +
                 "set title = ?, author = ?, genre = ? " +
                 "where id = ?";
    stmt = connection.prepareStatement(sql);
    stmt.setString(1, book.getTitle());
    stmt.setString(2, book.getAuthor());
    stmt.setString(3, book.getGenre());
    stmt.setInt(4, book.getID());

    if (stmt.executeUpdate() == 1)
        // OK
    else
        // ERROR
}
catch (SQLException e) {
    // ERROR
}
finally {
    if (stmt != null) stmt.close();
}
```

# Commit or Rollback the Transaction / Close the database connection

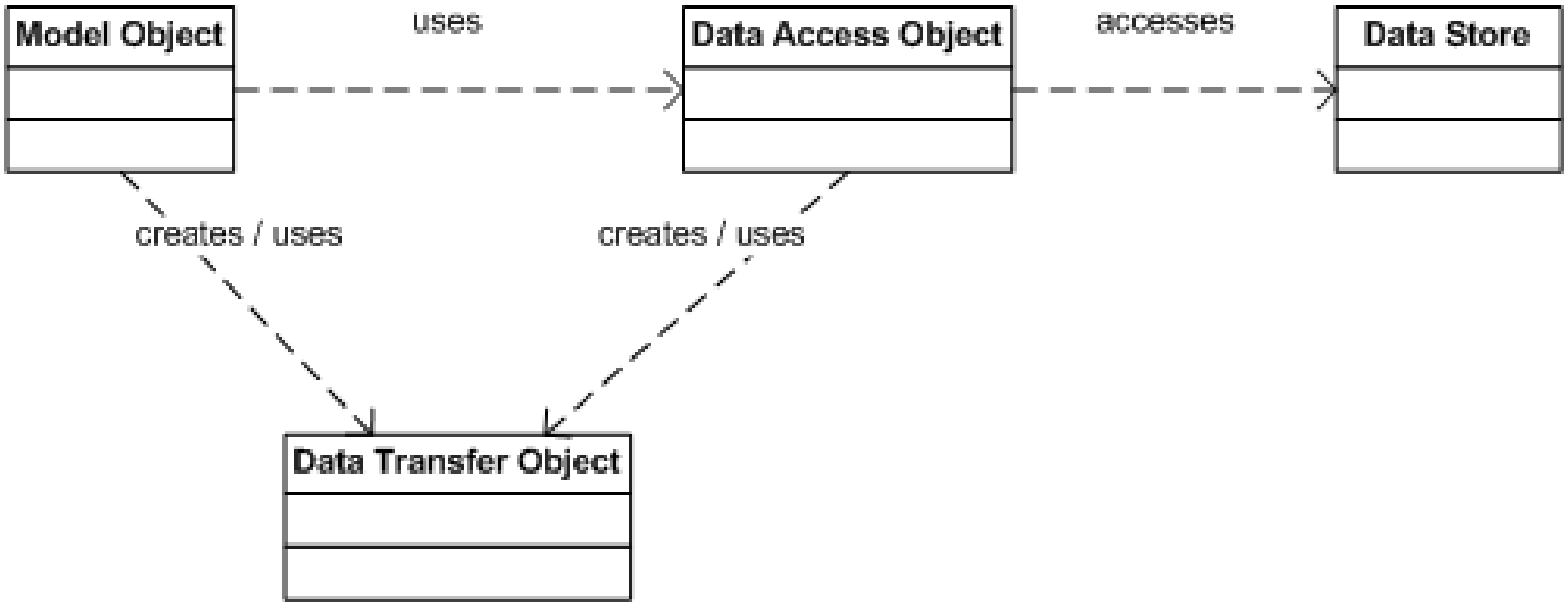
```
try {
    if (ALL DATABASE OPERATIONS SUCCEEDED) {
        connection.commit();
    }
    else {
        connection.rollback();
    }
}
catch (SQLException e) {
    // ERROR
}
finally {
    connection.close();
}

connection = null;
```

# Data Access Object Pattern

- Problem: Programs often need to use different data stores over time (or even at the same time). Achieving this is complicated by the fact that the APIs for different data stores vary.

# Data Access Object Pattern

- Solution: Isolate all code that interacts directly with the data store in "Data Access Object" (DAO) classes. All data store-specific code is encapsulated in DAOs. Any part of the program that needs to access the data store does so through the DAOs, thus isolating them from the details of the data store API. This structure makes it possible to support a new data store by modifying only the DAOs.



# Data Access Object Pattern

- Create "Data Transfer Object" (DTO) classes that can be used to shuttle data back and forth between the DAOs and other parts of the program.
- Classes in the core model have corresponding DTO classes
- DTOs are "relational" rather than "object-oriented"
  - They use keys to link objects rather than pointers
  - Pointers have no meaning in a data store, so keys are used instead

- Book Club Example

# Data Access Object Pattern

- Create "Data Access Object" classes that provide the CRUD operations required by the program
  - CRUD = Create, Read, Update, Delete

- Book Club Example

# Book Club Example

- TransactionManager
- Sequence Diagram

# Using Derby's `ij` console

- `set DERBY_HOME=C:\cs340\inventory-tracker\lib\derby-10.5`
- `set PATH=%DERBY_HOME%\bin;%PATH%`
- `ij`
- `connect 'jdbc:derby:db\inventory;create=false';`
- Run SQL commands
  - `SELECT * from product;`
  - `run 'createdb.sql';`
  - `...`
- `quit;`