Relational Databases

CS 240

Database Management Systems (DBMS)

- Databases are implemented by software systems called Database Management Systems (DBMS)

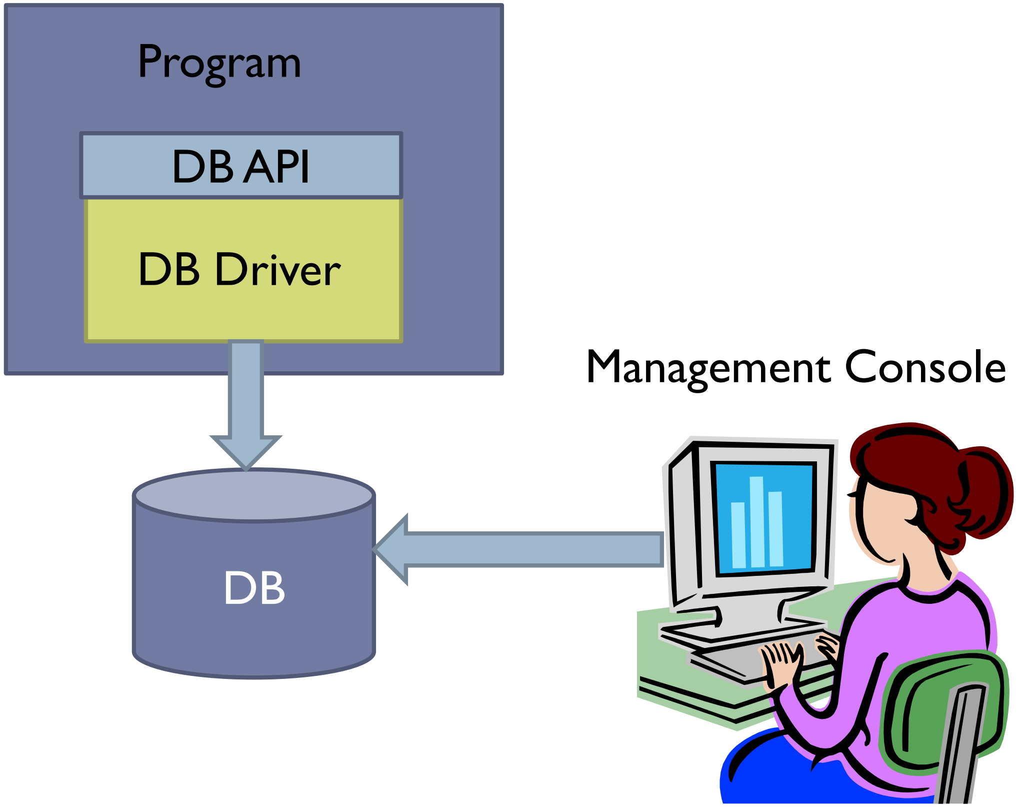
- Commonly used Relational DBMS's include MySQL, MS SQL Server, and Oracle

- DBMS's store data in files in a way that scales to large amounts of data and allows data to be accessed efficiently

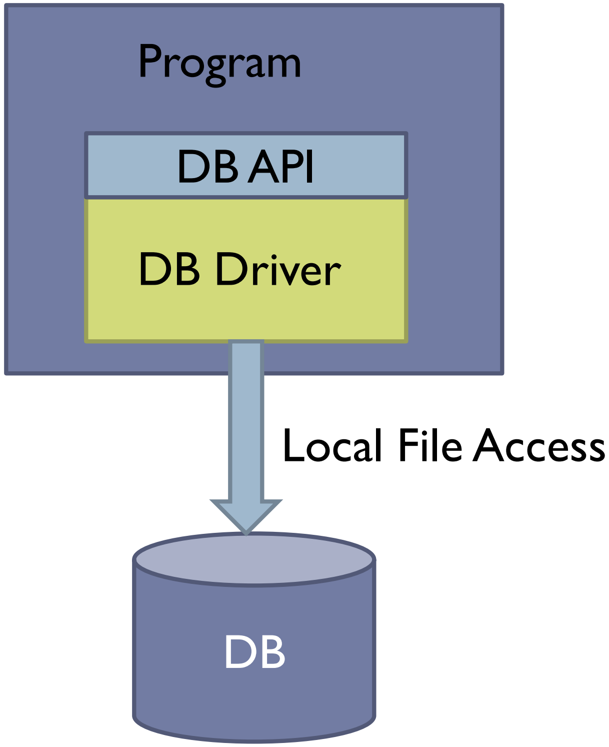
Programmatic vs. Interactive Database Access

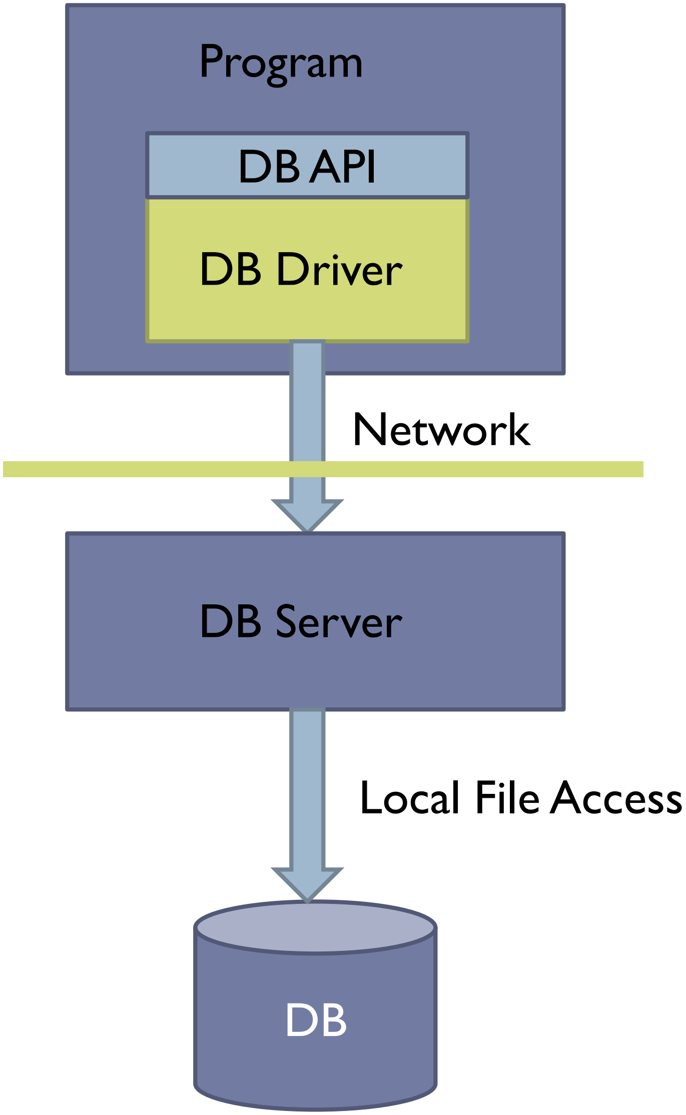
Programs can access a database through APIs such as ADO.NET or JDBC.

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 row 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

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

- Each column in an SQL table declares the type that column may contain.
- **Character strings**
- CHARACTER(*n*) or CHAR(*n*) — fixed-width *n*-character string, padded with spaces as needed
- CHARACTER VARYING(*n*) or VARCHAR(*n*) — variable-width string with a maximum size of *n* characters
- **Bit strings**
- BIT(*n*) — an array of *n* bits
- BIT VARYING(*n*) — an array of up to *n* bits

SQL Data Types

- **Numbers**
- INTEGER and SMALLINT
- FLOAT, REAL and DOUBLE PRECISION
- NUMERIC(*precision*, *scale*) or DECIMAL(*precision*, *scale*)
- **Large objects**
- BLOB – binary large object (images, sound, video, etc.)
- CLOB – character large object (text documents)

SQL Data Types

- **Date and time**
- DATE — for date values (e.g., 2011-05-03)
- TIME — for time values (e.g., 15:51:36). The granularity of the time value is usually a *tick* (100 nanoseconds).
- TIME WITH TIME ZONE or TIMETZ — the same as TIME, but including details about the time zone in question.
- TIMESTAMP — This is a DATE and a TIME put together in one variable (e.g., 2011-05-03 15:51:36).
- TIMESTAMP WITH TIME ZONE or TIMESTAMPTZ — the same as TIMESTAMP, but including details about the time zone in question.

SQLite Data Types

- SQLite stores all data using the following data types
  - INTEGER
  - REAL
  - TEXT
  - BLOB
- SQLite supports the standard SQL data types by mapping them onto the INTEGER, REAL, TEXT, and BLOB types

Creating and Deleting Tables

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

- DROP TABLE
  - Book Club Example

Modeling Object Relationships

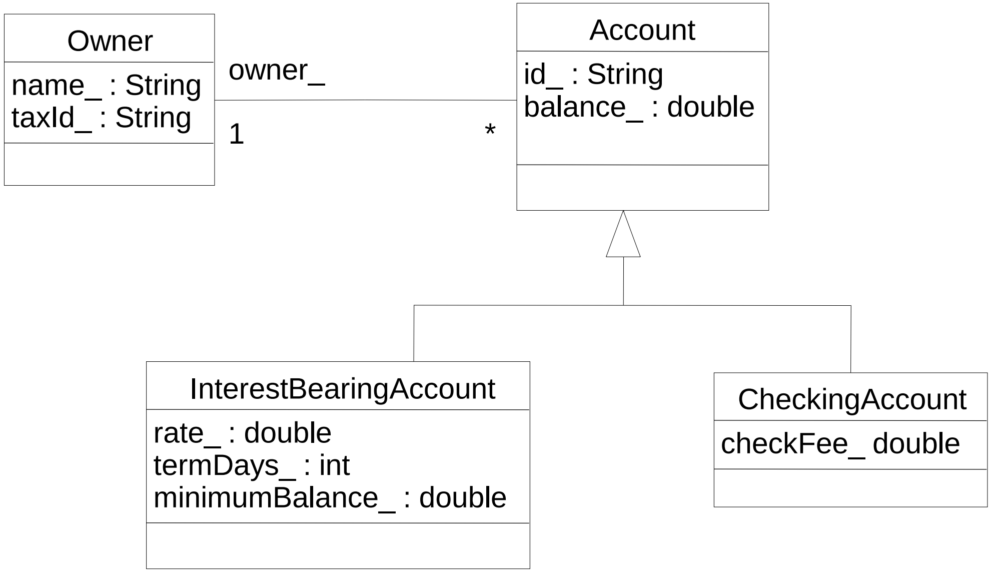
- Connections between objects are represented using foreign keys
- Foreign Key: A column in table T_1 stores primary keys of objects in table T_2
- Book Club Examples
  - Reading table stores Member and Book keys
  - Category table stores parent Category key
  - Category_Book table stores Category and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

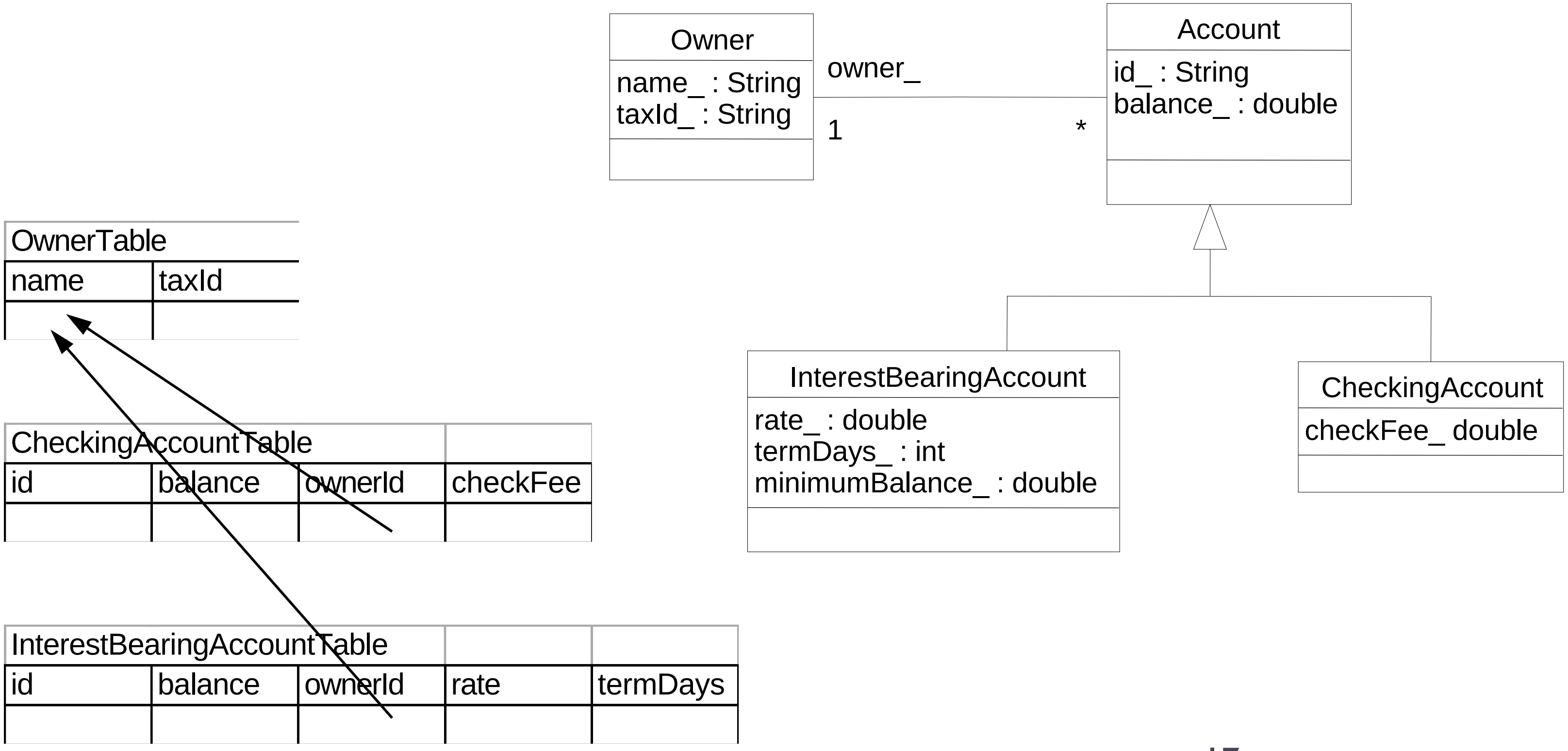
Modeling Inheritance Relationships

- How do we map the following Class Model to an RDBMS



Horizontal Partitioning

- Each concrete class is mapped to a table



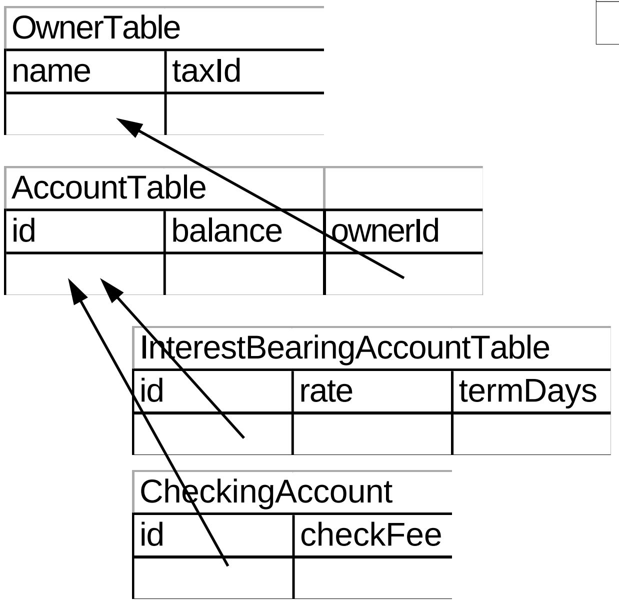
OwnerTable	
name	taxId

CheckingAccountTable			
id	balance	ownerId	checkFee

InterestBearingAccountTable				
id	balance	ownerId	rate	termDays

Vertical Partitioning

- Each class is mapped to a table

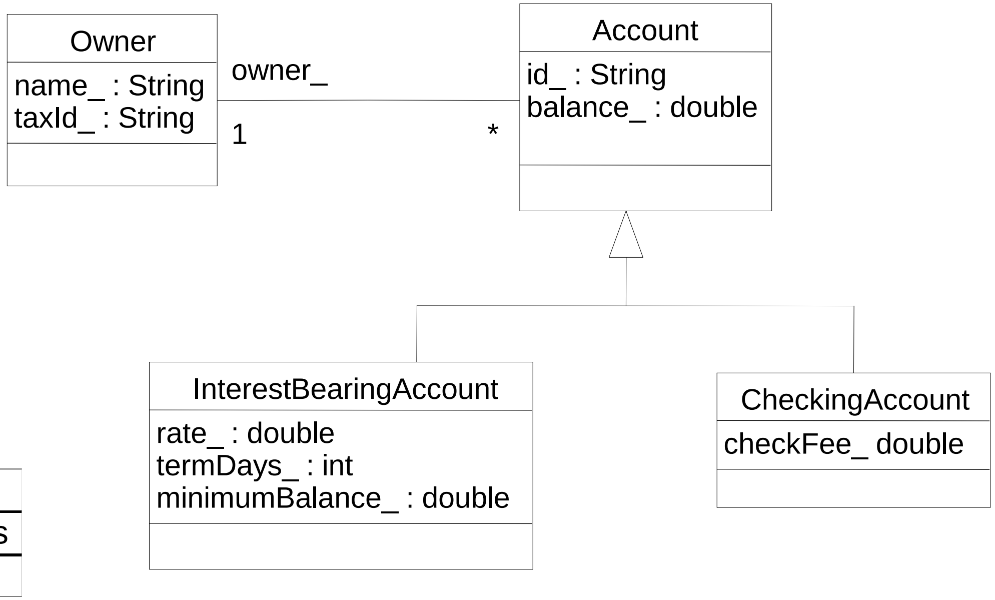


OwnerTable	
name	taxId

AccountTable		
id	balance	ownerId

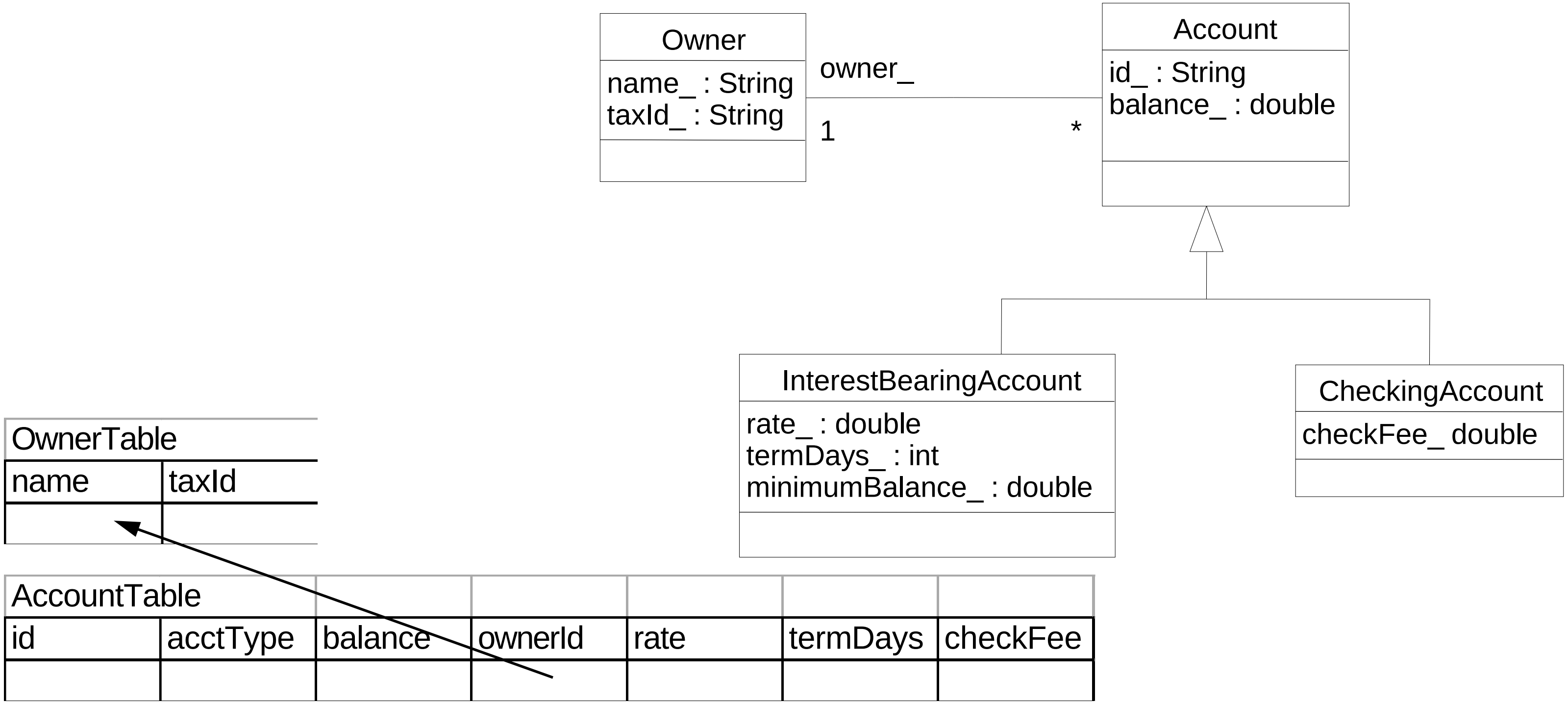
InterestBearingAccountTable		
id	rate	termDays

CheckingAccount	
id	checkFee



Unification

- Each sub-class is mapped to the same table



OwnerTable

name	taxId

AccountTable

id	acctType	balance	ownerId	rate	termDays	checkFee

RDBMS Mapping

- Horizontal Partitioning
  - entire object within one table
  - only one table required to activate object
  - no unnecessary fields in the table
  - must search over multiple tables for common properties
- Vertical Partitioning
  - object spread across different tables
  - must join several tables to activate object
- Vertical Partitioning (cont.)
  - no unnecessary fields in each table
  - only need to search over parent tables for common properties
- Unification
  - entire object within one table
  - only one table required to activate object
  - unnecessary fields in the table
  - all sub-types will be located in a search of the common table

Inserting Data into Tables

- INSERT
  - Book Club Example

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
```

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’

Database Transactions

- By default, each SQL statement is executed in a transaction by itself
- Transactions are most useful when they consist of multiple SQL statements, since you want to make sure that either all of them or none of them succeed
- For a multi-statement transaction,
  - BEGIN TRANSACTION;
  - SQL statement 1;
  - SQL statement 2;
  - …
  - COMMIT TRANSACTION; or ROLLBACK TRANSACTION;

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
    - All defined integrity constraints are enforced
  - I = Isolated
    - 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.
    - 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 - accessing a database from Java

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

- Retrieving auto-increment ids

Load Database Driver

```
import java.sql.*;

try {
       final String driver = "org.sqlite.JDBC";
       Class.forName(driver);
}
catch(ClassNotFoundException e) {
       // ERROR! Could not load database driver
}
```

Open a Database Connection / Start a Transaction

```
import java.sql.*;

String dbName = "db" + File.separator + "bookclub.sqlite";
String connectionURL = "jdbc:sqlite:" + dbName;

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 Insert, Update, or Delete

```
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;
```

Retrieving Auto-increment IDs

```
PreparedStatement stmt = null;
Statement keyStmt = null;
ResultSet keyRS = null;
try {
    String sql = "insert into book (title, author, genre) values (?, ?, ?)";
    stmt = connection.prepareStatement(sql);
    stmt.setString(1, book.getTitle());
    stmt.setString(2, book.getAuthor());
    stmt.setString(3, book.getGenre());

    if (stmt.executeUpdate() == 1) {
        keyStmt = connection.createStatement();
        keyRS = keyStmt.executeQuery("select last_insert_rowid()");
        keyRS.next();
        int id = keyRS.getInt(1);   // ID of the new book
        book.setID(id);
    }
    else
        // ERROR
}
catch (SQLException e) {
    // ERROR
}
finally {
    if (stmt != null) stmt.close();
    if (keyRS != null) keyRS.close();
    if (keyStmt != null) keyStmt.close();
}
```

Setting Up SQLite in Eclipse

- Use SQLite – already installed on the linux machines
- Download one of the following two SQLite JDBC drivers
  - sqlite-jdbc-3.7.2.jar
- Store it wherever you like

At Least Two Methods to Get it Working

- Both basically put the jar you just downloaded in the build path for your project.
- Technique 1:Right click on your project icon in the Package Explorer. In the menu select *Build Path* and then *Add External Archives*. Use the folder explorer that appears to find the jar file you downloaded and select "open" and it will be made part of your program's build path.

At Least Two Methods to Get it Working

- Technique 2:
  - Select ***Run*** at the top of the page.
  - Select ***Run Configurations…*** about 5 lines down.
  - Select the ***Classpath*** tab in the row of tabs underneath the name of your main routine.
  - In the Classpath window select ***User Entries***
  - Select Add External Jars… from the right column
  - Now navigate to the folder where you stored your sqlite jdbc jar file
  - Select the jar file
  - Hit the ***Open*** button
  - Then select **Apply** button

Installing SQLite3 on Linux

- Linux
  - Download the source file from (usually the second file listed) http://www.sqlite.org/download.html
  - tar –xzvf the downloaded file
  - cd to the new folder
  - ./configure
  - make
  - make install

Installing SQLite3 on a Mac

- On a recent OS you don’t have to, it is already there

Installing SQLite3 on Windows

- Download the first two zip files from the section labeled **Precompiled Binaries for Windows**.
- Unzip them and place the three resulting files in C:\WINDOWS\system32 (or any directory on you PATH.
  - Alternative: I created a new directory called SQLite in C:\Program Files (x86) and placed the three files in that location. I then extended the PATH variable to search that location

Adding the SQLite Manager to Firefox

- You can manage an SQLite database using the command line and text-based SQLite commands, but, it is easier to the SQLite Manager extension you can get for Firefox.
- First, start Firefox
- Then go to

  https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/

  and hit the green "Add to Firefox" button and install the extension.
- After it is installed you can click on the "SQLite Manager" under the Tools tab at the very top.