

Course Information

Welcome to the blended learning course "Modern Database Techniques". Here you will find some information about the course: content, target group, learning objectives, and course structure.

For all exercises we use the object oriented database management system Caché from [InterSystems](#). Here is a link to the [Caché documentation](#) of InterSystems. You will find links to parts of the documentation also within the learning modules.

The Caché database can be [downloaded](#) for free as single-user version. First, you have to register and then you'll be able to download and install Caché. Or you can access the database server of the University of Applied Sciences Ravensburg-Weingarten. You will get instructions how to do that in the first classroom lesson. This is also described in the learning module "Concepts of Object Oriented Databases".

Target Group

This course is for students of a master in Computer Science or bachelor in 5th semester or higher.

Prerequisites are:

- Relational databases and SQL
- Object oriented programming
- Software engineering

Learning Objectives

After this course participants ...

- know the difference between Relational Database Systems and Object Oriented Database Systems. They can choose the appropriate system for a given problem.
- can design the structure of an object oriented database using UML
- know the theory of object oriented database systems and how the theoretical concepts are implemented in the postrelational database management system Caché
- can develop a server application using Caché and the programming language Object Script
- can develop a client application in Java using the Java Language Binding
- know the internal data structure of Caché: Globals, Indexes, and B-trees
- can install and administrate Caché DBMS

- know the mechanisms for transactions, locking, crash recovery and backup in Caché
- can write correct software for multi-user databases using their knowledge of different locking mechanisms of Caché
- know different security mechanisms and which of them are available in Caché

Content and Course Structure

Learning module 0: **Course Information, Welcome** 1 h (auf Wunsch in deutscher Sprache)

- Welcome and getting acquainted (Begrüßung und Kennenlernen)
- Course organization (Kursorganisation)
- Installation of Caché (Installation von Caché und Jalapeño)

Learning module 1: **Introduction to Object Oriented Databases:** 4 h

- Review of Object Oriented Programming Concepts
- Sample Application: Car Rental System (CRS) with Exercise: Relational implementation of sample application
- Different object oriented database technologies

Classroom lesson (Präsenztermin) 1: **Caché und Jalapeño:** 8 h (Termin wird vereinbart, Präsenzveranstaltung auf Wunsch in deutscher Sprache)

- Discussion of the results of the first exercise (Diskussion der Ergebnisse der ersten Übung)
- Identify advantages and disadvantages of relational model (Vor- und Nachteile des relationalen Datenbank-Modells)
- Development environment for Caché Jalapeño (Arbeiten mit der Entwicklungsumgebung Eclipse für Jalapeño)
- Basics of Jalapeño (Grundlagen von Jalapeño)
- Building of learning teams with two students each (Lernteams zusammenstellen)
- Introduction to test driven development and pair programming (Programmiertechnik Paar-Programmierung)

Content and Course Structure (2)

Learning module 2: **Concepts of Object Oriented Databases:** 40 h

- Type constructors, complex objects
- Classes: Type- and Set-View
- Relations between Classes
- Object Identity
- Inheritance
- Query Operations
- Object Query Language (OQL) of ODMG
- Methods and Polymorphism
- Integrity

- Trigger
- All topics are practised developing a server application with Caché for the CRS example

Learning module 3: **Client Application and Language Binding: 20 h**

- Java Language Binding
- Avoiding the object-relational mismatch
- Development of the client application for the CRS example

Classroom lesson (Präsenztermin) 2: **Discussion of problems and solutions of the exercises 4 h** (Termin wird vereinbart, auf Wunsch in Deutsch)

- Discussion of problems with the concepts of OODBMS, Caché, and programming the CRS application (Fragen der Teilnehmer)
- The next learning modules (Übersicht über die folgenden Lernmodule)

Content and Course Structure (3)

Learning module 4: **Advanced storage concepts, performance tuning: 10 h**

- Storage organization: globals, MAP blocks, and b-trees
- Indices: standard index, bitmap index, and bitslice index

Learning module 5: **System Management: 20 h**

- Transactions and Locks
- DBMS read and write processes
- Log-files = Journals and Crash Resiliency
- Caché Backup
- Shadowing

Learning module 6: **Security in Databases: 10 h**

- Authentication: Who may use database
- Authorization: Which data may a user access
- Encryption: Additional security
- Auditing: Track of DB access

Learning module 7: **Distributed and Mobile Databases: 25 h**

- Different kinds of distributed database systems
- Enterprise Cache Protocol (ECP) in Caché
- Synchronisation and coherency in distributed systems
- Transactions in distributed systems
- Mobile databases

Web conference: **Exam Preparation: 2 h**



Hochschule
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

Modern Database Techniques

Course Introduction

**Concepts of Object Oriented
Programming Revisited**



Object oriented Concepts: Overview

- Classes and objects
- Inheritance and class hierarchies
- Aggregation, compound objects
- Methods
- Object identity
- Abstract data types



Classes and Objects: Example in C++

- Similar objects form a class

```
class Fraction
{private: long Numerator;
         long Denominator;
         reduce();

public:   Fraction (long n, long d = 1)
         {Numerator = n; Denominator = d; reduce();}

         Fraction& plus(const Fraction &x)
         {Numerator = Numerator * x.Denominator + Denom...
          Denominator = Denominator * x.Denominator;
          reduce();}

         void print ()
         {cout << Numerator << "/" << Denominator;}
}; // End of class definition
```



Objects instead of Variables

- In the following code example two fraction objects f1 and f2 are constructed. Then they get messages to perform the methods plus and print.

```
Fracti on f1 (3, 4);  
Fracti on f2 (1, 7);  
f1. pl us(f2);  
f1. pri nt ();
```

- 25/28 is printed.
- Classes don't form a database:
"Search for all objects of class Fraction > 1/4"
is **not** possible.
- The objects are not persistent:
They only exist during program execution.



Inheritance: Classes with Subclasses

```
class Point
{private: int x, y;
 public:
   Point (int px = 0, int py = 0) {x=px; y=py;}
   move (int dx, int dy) {x+=dx; y+=dy;}
   int draw ();
}
```

```
class Rectangle: public Point
{private:
   int height, width;
 public:
   Rectangle (int x, int y, int height, int width);
   int draw ();
}
```

Rectangle is a subclass of Point. In addition to the attributes x and y a rectangle has the attributes width and height.



Aggregation

- Objects may consist of component objects

```
class Line
{private:
    Point start, end;
    int linewidth;
public:
    Line (int xs, int ys, int xe, int ye, int lw);
    int draw ();
}
```

- *Rectangle* is subclass of *Point*, i. e. the lower left point and additionally height and width.
- *Line* is not a subclass of *Point*, but is an aggregation of initial point and end-point.



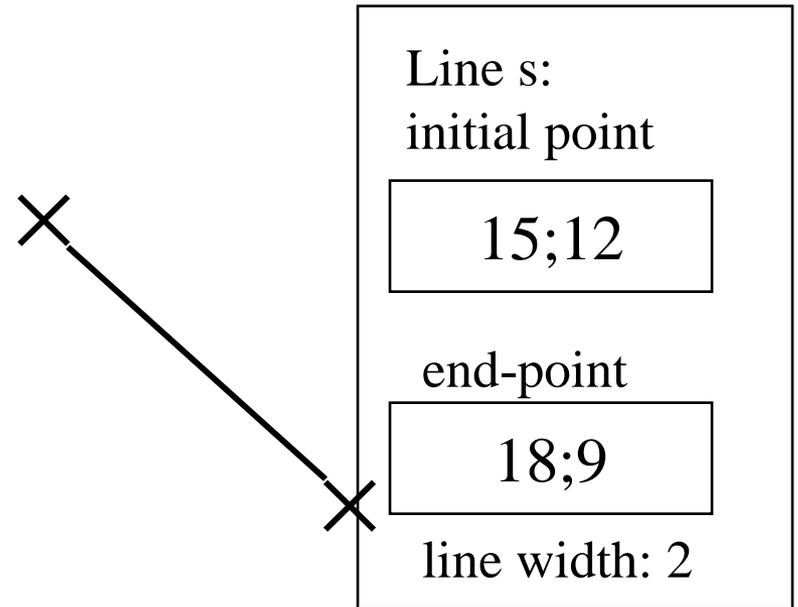
Comparison of Subclasses and Aggregation

✗ Point a: 2; 15



a special point with height and width

Rectangle w: 2; 3;	5; 3
--------------------	------



correct: a.x, w.x
s.end.x

wrong: s.x



Methods

- Methods are functions belonging to a class.
 - Instance methods: Let a be an object and m be a method of class c . $a.m(\text{parameter})$ sends a message to a , to run method m .
 - Class methods are independent of instances, e. g. constructor methods.
- Advantages:
 - Less side effects
 - It is similar to get an attribute or to call a method
→ it's easy to modify a method's implementation, e. g.
 - Attribute Age or
 - Method Age: calculate age using the date of birth
 - Interface definition and implementation are separated



Object Identity

- Each object gets a unique identifier: OID
- This OID does not change during the whole lifetime of the object.
- The OID remains the same even when the object's values change.
- OIDs need not be specified explicitly.
- OID can't be changed.



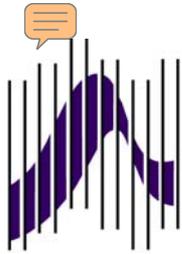
Classes Implement Abstract Data Types (ADT)

- Abstract data types define the behavior of data, but hide the implementation.
- Classes are suitable to define ADTs.
- Especially template classes are used.
- Example: ADT Stack
Use a template class Stack to generate:
 - stacks for integer numbers
 - stacks for floating point numbers
 - stacks for Persons



Example ADT Stack

```
TYPE:          Stack [e_Type]
METHODS:       empty?:      Stack [e_Type] → BOOLEAN
               new:         Stack [e_Type]
               push:        e_Type × Stack [e_Type]
                               → Stack [e_Type]
               pop:         Stack [e_Type] → Stack [e_Type]
               top:         Stack [e_Type] → e_Type
CONSTRAINTS:  pop (k ∈ Stack[e_Type]): NOT empty? (k)
               top (k ∈ Stack[e_Type]): NOT empty? (k)
RULES:        ∀e ∈ e_Type, k ∈ Stack[e_Type]:
               empty? (New());
               NOT empty? (push (e, k));
               top (push (e, k)) = e;
               pop (push (e, k)) = k.
```



Hochschule
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

Modern Database Techniques

Classroom Lesson 1



Classroom Lesson 1 and 2: Overview

- Welcome
- Introduction of the Participants
- Course Organisation
- MOODLE 
- CRS Example
- Discussion: CRS Example in Relational Model
- Advantages and Disadvantages of Relational Model
- Installation of Caché
- Access to Neptun-Server
- Eclipse and Jalapeño
- Managing Caché
- Test Driven Development and Pair Programming
- Learning Teams
- Presentation of Various Object Oriented Database Technologies



CRS Example

- Lecture and exercises based upon an application:
Car Rental System (CRS) to rent cars at a Car Rental Company (CRC).
- A car rental company has many Branches.
- Branches provide various vehicles.
- There are different types of vehicles: cars, trucks, and motor cycles.
- Customers rent vehicles.
- Reservations may be prepared online.



Sample Scenario: Use Case Reservation

- A customer comes to a branch.
- One of the employees of the CRC attends to the customer.
- The customer wants to reserve or rent a car.
- The employee starts the CRS-application on his computer.
- If customer is not yet registered he first will be registered.
- The customer provides information which vehicle he wants to rent and when.
- A search form of the CRS is filled with this information. CRS shows list of available vehicles.
- The customer gets detailed information about each vehicle by clicking on a list item. At last he chooses one vehicle.
- At this time the CRS locks this vehicle to avoid conflicts.
- Check driving license of the customer. If it is not sufficient cancel the process.
- The reservation is stored in the database and a contract is signed. Unlocked vehicle in database!
- If reservation date = actual date goto delivery process.



Sample Scenario

Object Types	Description
Branches of the CRC	Name, Location, Opening Hours, Phone Number, Provided vehicles
Model	Manufacturer, Type, Subclass
Vehicles	Licence Number, Power, Fuel, Seats, PriceKm, PriceDay,
Cars, Trucks, Motor Cycles	special vehicles
Persons	Name, Date of Birth, Email, List of Phone Numbers (Number, Type), List of Addresses (State, City, Street or PO Box, ZIP, Type)
Employees of CRC	Like Persons, additionally Task, Salary, SSN, Branch, Supervisor, ... Relation: Employees advise Customers



Sample Scenario (Cont.)

Object Types	Description
Addresses	Country, State, City, Street or PO Box, Zip, Type
Customers of the CRC	Customers can be Persons or Companies CustomerNr, CostomerSince, Discount
Persons as Customers	Like Persons and Customers, Relation to Driving Licenses
Companies as Customers	Like Customers, additionally Name, Set of Representatives (Persons)
Driving License Classes	Abbreviation, Description
Loans	start date, end date, start km, end km, rented vehicle, status

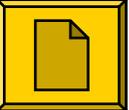


CRS Example

Your Relational Solution



Relational Implementation: Difficulties (1)

- Difficulties with Name of persons
 - One column → difficult search and ordering
 - Columns for First Name, Last Name, Title, ... → no connection between the names' parts
 - Extra table for Names → complicated structure
- Difficulties with Phone Numbers (Addresses) 
 - One column with one string, numbers separated by a separator (; or |) → difficult search (LIKE)
 - Extra rows for each number → redundancy
 - Extra table (1st normal form) → complicated structure, bad performance



Relational Model: Disadvantages

- Addresses and Phone Numbers of one Person not in one table.
- Information of one Person distributed.
- Keys, foreign keys and integrity constraints necessary to maintain consistency.
- Natural keys are often long → redundancy
→ artificial keys must be used
- Connection between structured attributes not evident



Conclusion 1



Attributes,
consisting of value sets or
of many components,
can be simulated,
but not directly represented.
Physical data independence violated



Relational Implementation: Difficulties (2)

Problem: Employees are persons

Possible solutions

- Table Person with additional columns for attributes of employee and Person Type →
Many NULL-values
- Employees in table Employee, other persons in table Person →
Loss of information: employees are also persons
Table Address has to reference two tables
- Table Employee has only additional columns, relation to table Person by foreign key →
Join necessary to get all data of an employee



Relational Implementation: Difficulties (3)

Problem: Supervisor of employees

Possible solution

- Relation from table Employee to table Employee by a foreign key



Conclusion 2



Foreign keys have different meanings:

- Multi valued attributes of an object-type
e.g. Address.PNr \rightarrow Person.PNr
- Specialization of an object-type
e.g. Employee.PNr \rightarrow Person.PNr
- components of an object-type
e.g. Employee.Super-PNr \rightarrow Employee.PNr

The different meanings cannot be distinguished.



Relational Implementation: Difficulties (4)

Employee Advises Customer

- Many-to-many relationship
- To avoid redundancy and anomalies transform into Boyce-Codd Normal Form
- Create connection-table "Advises" with foreign keys to Employee and Customer
- Disadvantage:
 - Customer wants to see a list of employees who advise him
 - Employee wants to see a list of customers he advises
 - Join operations necessary to construct these lists



Queries using SQL (1)



- Practice: Define an SQL-query which gives all information of an Employee:
 - Name, addresses, phone numbers, date of birth,
 - Position, Salary, SSN, ...
 - Supervisor (with phone numbers),
 - List of Tasks
- How clear is the result of the query
- What happens if the information contains NULL-values?



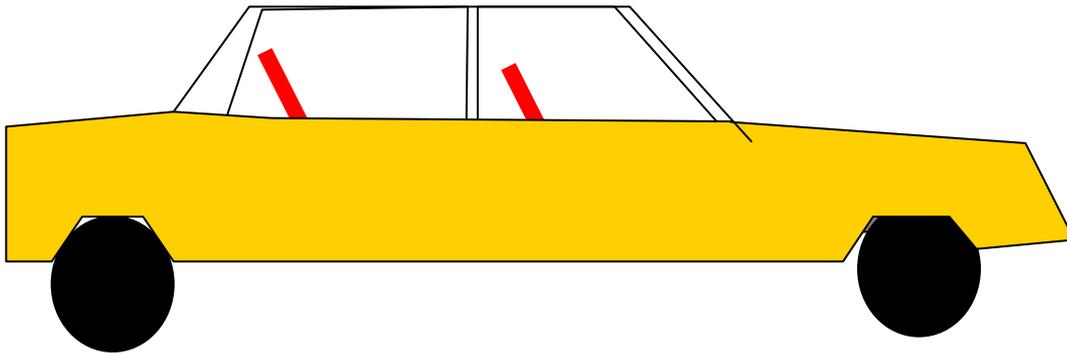
Queries (1): Result

```
SELECT *
FROM Employee E, Person PE, Employee SE,
     Person PS, Task T, EmpHasTask ET,
     Address EA, Phone EP, Phone SP
WHERE E.PNr = PE.PNr
      AND E.PNr = EA.PNr
      AND E.PNr = EP.PNr
      AND E.Super_PNr = SE.PNr
      AND SE.PNr = PS.PNr
      AND PS.PNr = SP.PNr
      AND E.PNr = ET.PNr
      AND ET.TNr = T.TNr;
```



Queries (1): Disadvantages

- Query must join all artificially scattered information.
- Virtual copies of the tables are needed (e.g. Person PE, Person PS).
- Complicated foreign key and join conditions necessary.
- NULL-Values: e.g. an employee has no supervisor → this employee is not in the result.
- Remedy: more complicated Outer-Joins
- Result is difficult to read due to redundant information → extra program or report necessary





Queries (2)

- Find all persons (only PNr) with the same work-address as the person with PNr = 2
- ```
SELECT PNr
FROM Address A1
WHERE A1.Type = 'work' AND
 (A1.ZIP, A1.City, A1.Street, A1.State) =
 (SELECT A2.ZIP, A2.City, A2.Street, A2.State
 FROM Address A2
 WHERE A2.PNr = 2 AND A2.Type = 'work');
```
- Disadvantage: SQL does not support structured attributes.



## Queries (3)



- Practice: Find an employee (only PNr) who has the tasks purchase (Tnr = 1) and billing (Tnr = 3)
- Wrong solution:  

```
SELECT EmpNr FROM EmpHasTask
WHERE Tnr = 1 AND Tnr = 3;
```
- (One) correct Solution:  

```
(SELECT EmpNr FROM EmpHasTask
WHERE Tnr = 1) INTERSECT
(SELECT EmpNr FROM EmpHasTask
WHERE Tnr = 3);
```



## Queries (3): Disadvantages

---

- Reason for wrong solution:
  - You think about an employee as an object.
  - This object has as property the set of his tasks.



## Queries (4)

---

- Find all persons with the same phone numbers as John Smith (PNr = 1)
- Wrong solution:  
SELECT PNr, FirstName, LastName  
FROM Person WHERE  
    (SELECT PhoneNr From Phone  
      WHERE Phone.PNr = 1) =  
    (SELECT PhoneNr From Phone  
      WHERE Phone.PNr = Person.PNr)
- Not allowed in SQL



## Queries (4): Disadvantages

---

- In SQL there are good, bad and not further usable tables.
- `SELECT ... FROM ... WHERE` defines a table, but a **bad** table
- It may be used in a query's `WHERE`-part as a sub-query, but not in the `FROM`-part.
- Comparison of two sets (relations, tables) in the `WHERE`-part like `'='` or `'<'` is not possible
- A `UNION`-query is again a table, but this table can't be further used.



# SQL is not a complete programming language

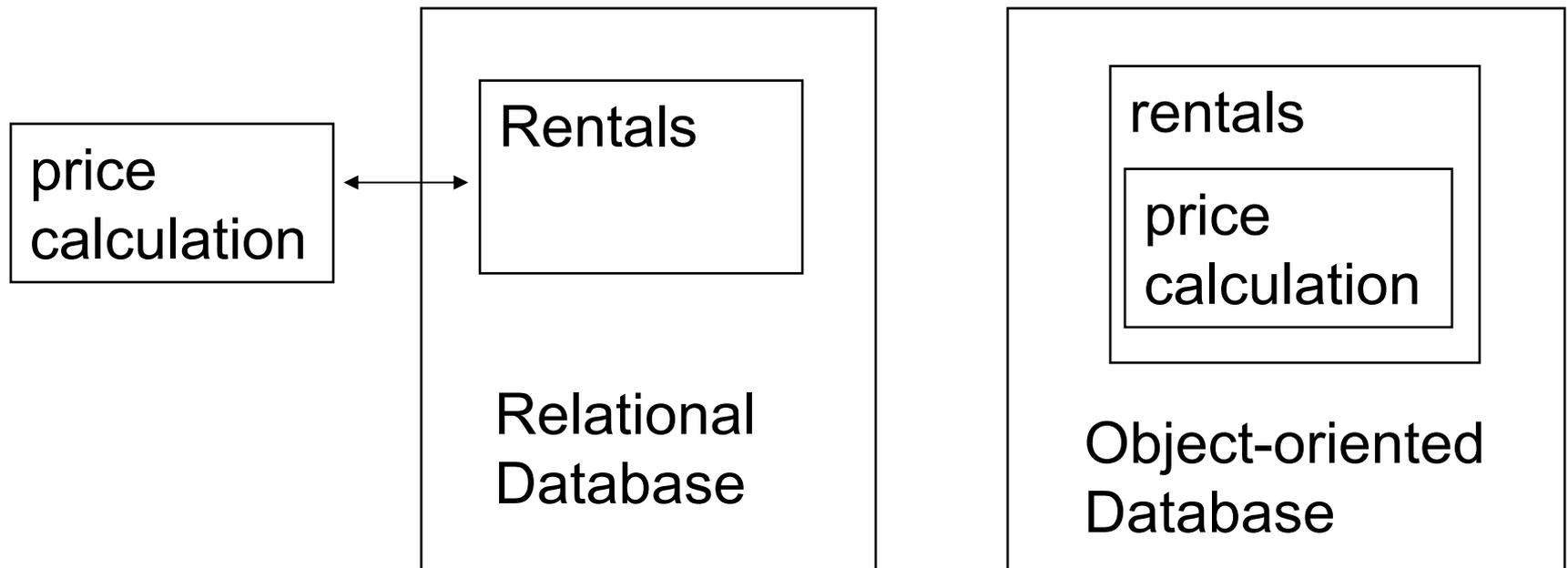
---

- Tasks of SQL-DML: insert, update, delete and search for data
- SQL cannot do
  - complex computations
  - Conditional statements
  - Loop statements
  - Function definition and recursion
- To perform complex tasks like price calculation for rentals embed SQL in programming languages



# Disadvantages application development

- Application programs even if part of the data model must be developed separately.
- Parts of the data models semantic is separated, database is not aware of that.





## Disadvantages application development (2)

---

- Object relational mismatch:
  - Data-types of databases and programming languages don't fit
  - Problems with NULL-values in databases: indicator variables necessary.
  - Set results of SQL-queries and single value variables of programming languages don't fit.
- Using OODBs, application programming and database access build a unit.



## Data modification (Insert, Update, Delete)

---

- Example: new person as customer (with 2 addresses, 2 phone numbers and 1 driving license)
- 7 rows must be inserted into 5 tables.
- Integrity constraints must check foreign keys → decrease of performance
- In OODBs **one** new object with all its properties is created.



## Modification of key values

---

- Keys can change e. g. license plate of a car when sold.
- All corresponding foreign keys must be changed in the same transaction.
- Many rows must be locked.



## Problem: Mutual foreign keys

---

- Example: Every employee has the branch number he works at as foreign key;  
Every branch has the PNr of its manager as foreign key.
- Insert a new branch but manager does not yet exist in DB → foreign key constraint violated
- Insert a new manager but branch does not yet exist in DB → foreign key constraint violated



## Conclusion 3 (SQL)



### SQL is a powerful language

- for searching data
- for manipulating many data with one statement

### Disadvantages

- Concept of tables defined by SELECT not consistent  
(disadvantage of SQL, not of relational concept)
- Complex, complicated statements due to scattered information
- Connection to other languages necessary
- Application and database development separated



# Conclusion: Relational Databases



## Advantages

- One simple model even for most complex applications
- Simple mathematical theory: relational calculus
- If front-end is not necessary relational databases can be built with little programming effort

## Disadvantages

- Information is scattered
- Bad performance for complex queries
- Natural object view of world not well supported
- Difficult application development



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques

## Part 1: Object Oriented Databases

---

### **3. Different Kinds of OODB**



# Overview

---

- OO language + persistence
- Pure OODB
- DB with object and relational access
- Object-Relational Databases
- Object oriented interface to relational databases



## OO language + persistence

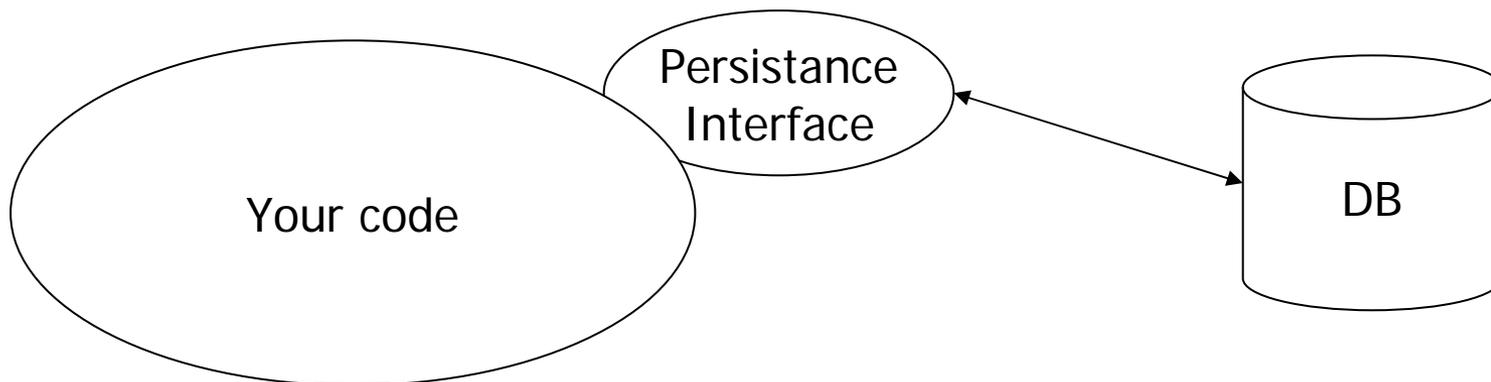
---

- Use a class library with an abstract class or interface for persistent objects.
- In your application create persistent classes as subclasses of that abstract class.
- Search for objects by object navigation. Write code for search yourself.
- No additional software like a database browser provided.
- Often only one OO language supported. No access to data by other languages.



# OO language + persistence

- Dependent on the special product more or less support for transactions or concurrent access provided.
- Advantage: Not much overhead
- Disadvantage: Not much functionality
- Examples: GemStone, ozone





# Pure OODBMS

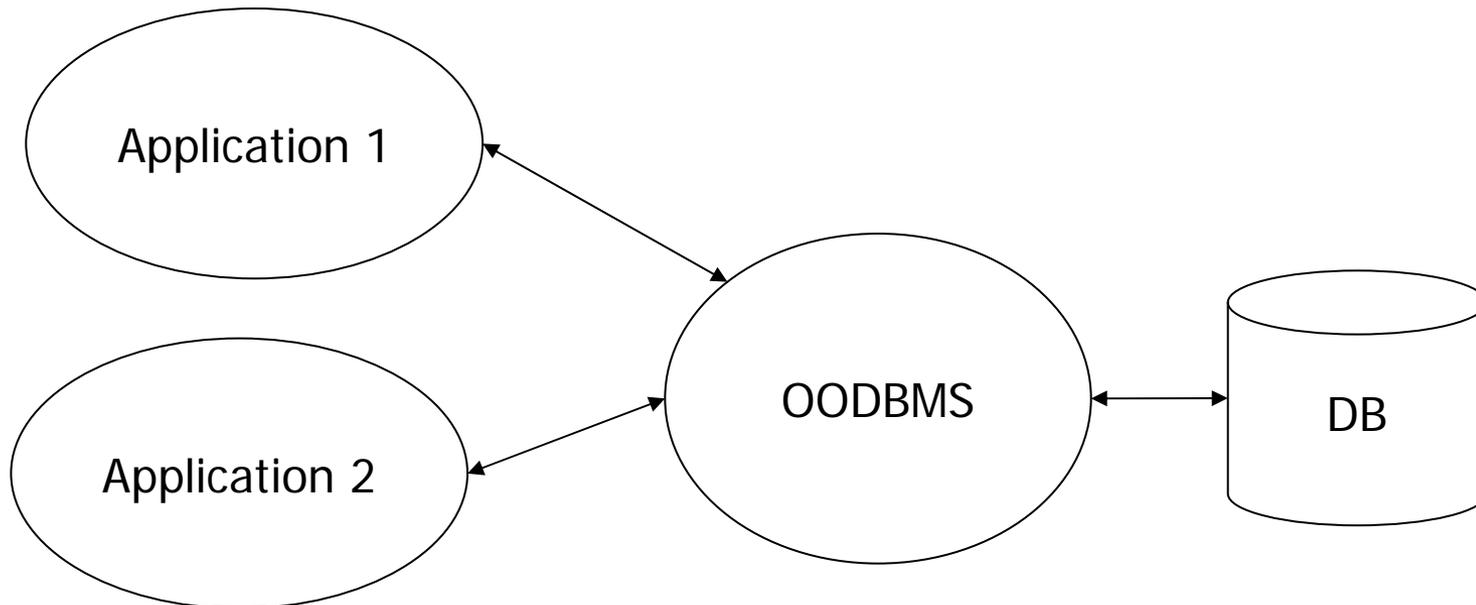
---

- Pure OODBMS are similar to "OO language + persistence"
- They enhance this model considerably:
  - OODBMS implement a subset of the ODMG-model
  - Variety of predefined template classes for sets, lists, arrays, ... with appropriate functions
  - Persistent class offers a query-function
  - Transactions, concurrency, logging, recovery, constraints, and triggers are supported.
  - Tools for DB-schema development
  - Tools like DB-browser



## Pure OODBMS (cont.)

- Advantage: Complete DB functionality
- Disadvantage: Often only one or two OO languages supported.
- Examples: ObjectStore, Versant





## DB with object and relational access

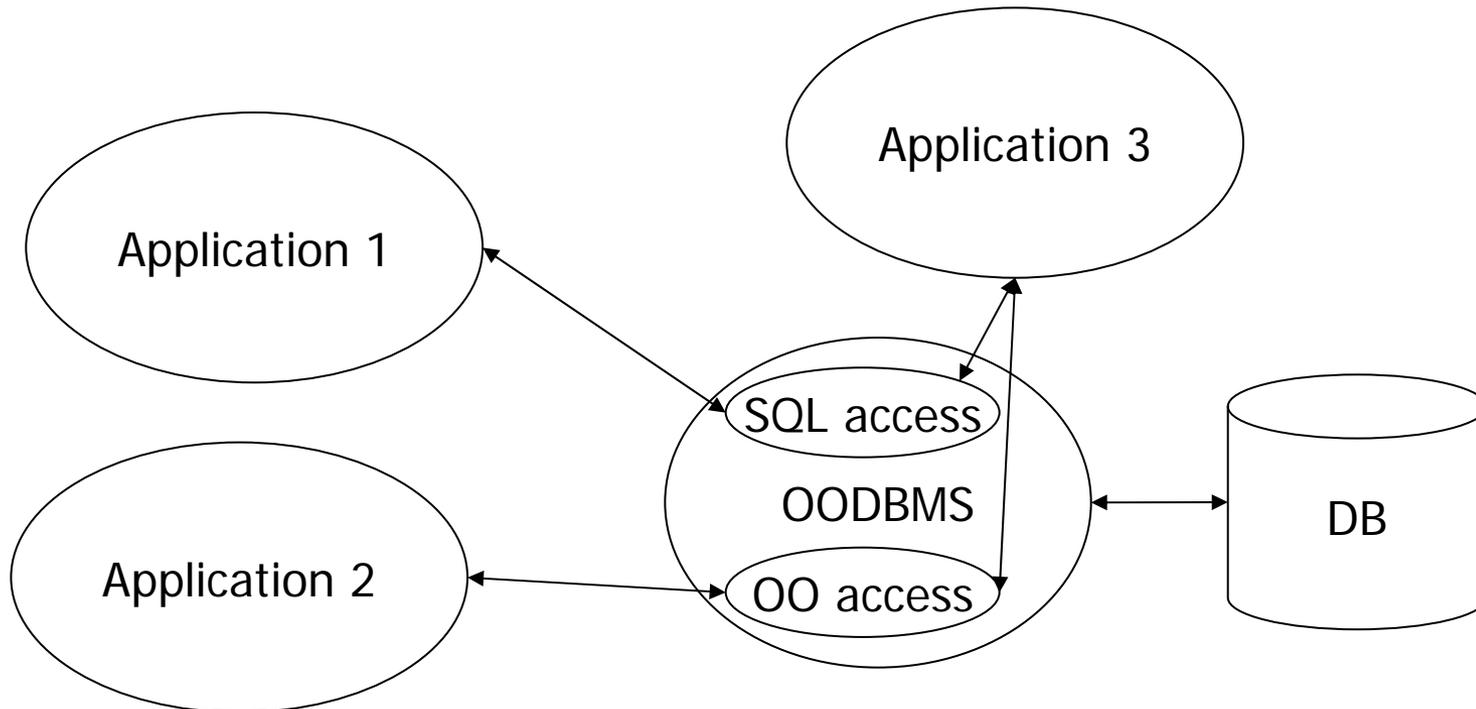
---

- Like pure OODBMS **and** relational databases
- Two different interfaces to database:  
OO interface **and** SQL interface
- SQL is also used for DB queries on the object side.
- Advantages:
  - Well known query language
  - Support of standard interfaces e. g. ODBC and JDBC
- Disadvantage: Not ODMG compliant.



# DB with object and relational access

- Examples:
  - Caché (InterSystems)
  - Titanium (Savitar Corp.)





# Object-Relational Databases

---

- Object oriented technology on top of relational technology and in the relational context.
- Define types instead of classes. Objects are stored in tables of objects rather than in tables of rows.
- Support of major object oriented features: complex types, inheritance, aggregation, methods
- Advantage: Extension of a well known technology
- Disadvantages:
  - Mixture of both technologies may result in difficult to understand schemas
  - Performance problems



# Object-Tables instead of Tuple-Tables

## Object-Table Persons

|                      |
|----------------------|
| Meier, 1.5.60, ...   |
| Huber, 11.5.69, ...  |
| Miller, 1.12.60, ... |
| Meier, 7.5.80, ...   |

Person\_type:  
Name,  
DateOfBirth,  
...

## Tuple-Table Persons

| Name   | DateOfBirth | ... |
|--------|-------------|-----|
| Meier  | 1.5.60      |     |
| Huber  | 11.5.69     |     |
| Miller | 1.12.60     |     |
| Meier  | 7.5.80      |     |

Separation of structure definition  
and table definition as set of  
objects instead of set of tuples



# Object-Relational Databases

---

- Examples:
  - Oracle 10g
  - IBM DB2
  - PostgreSQL



# Object Oriented Interface to Relational Databases

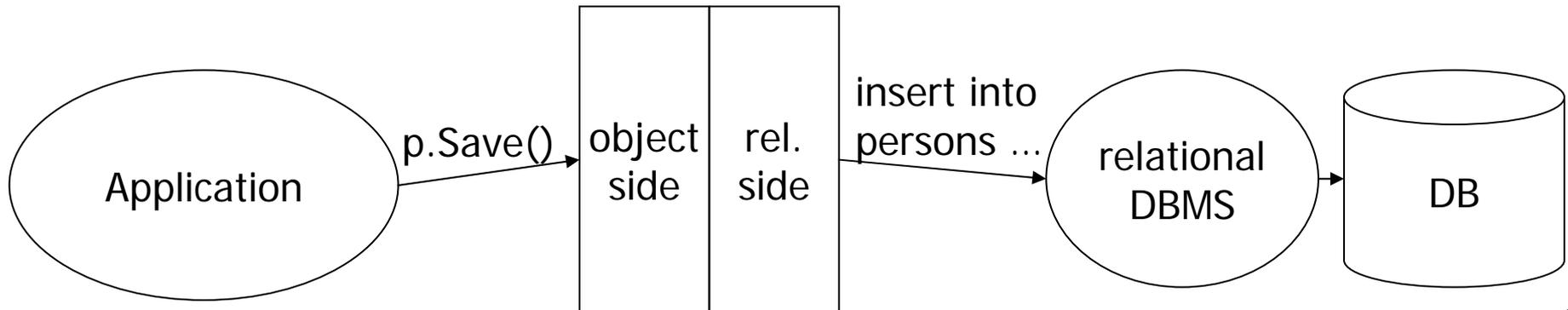
---

- Object oriented access to persistent objects
- Persistent objects stored as rows in relational database
- Mapping of objects to rows transparent to application



# Object Oriented Interface to Relational Databases

- Advantage: Use familiar object oriented programming techniques for your applications and relational techniques for databases.
- Disadvantages:
  - Mapping not fully transparent
  - Performance problems
- Example: Hibernate





Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Working with Caché

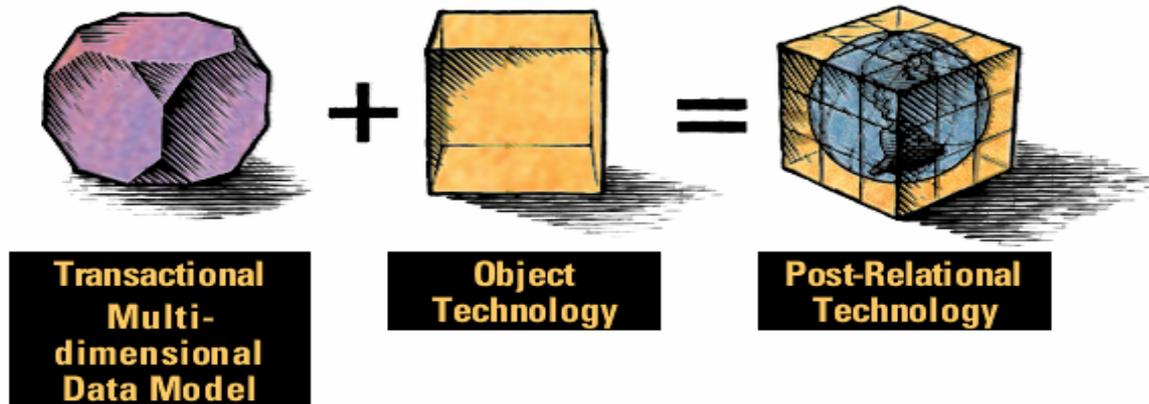
---

Introduction to  
Caché Jalapeño  
and Caché Objects



# Cache

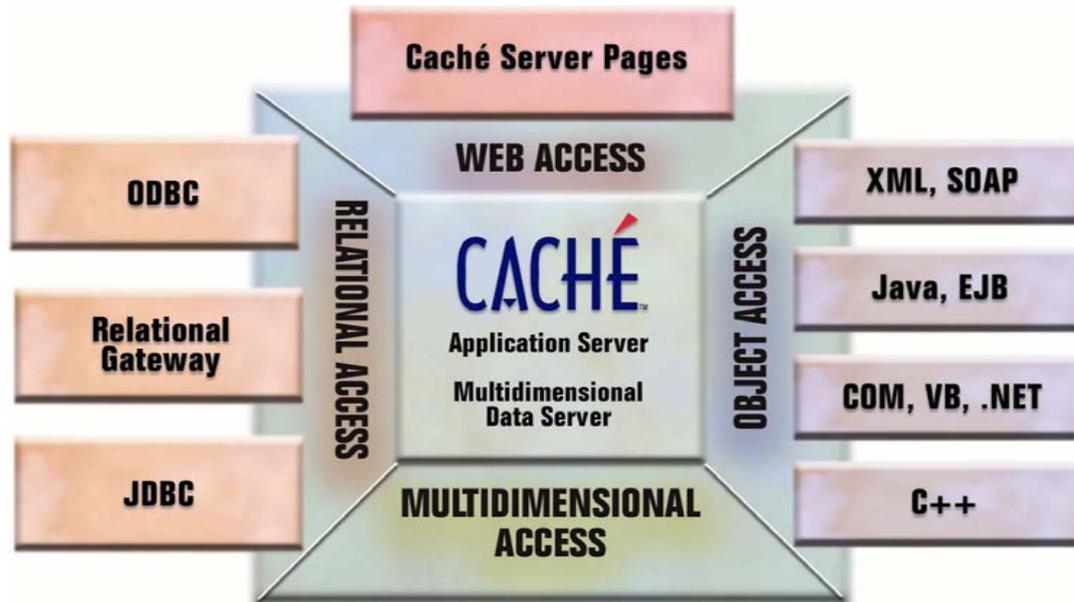
- High performance, massively scalable, highly secure, post-relational database management system.
- Integrated object-oriented and SQL access (Unified Data Architecture) to efficient multidimensional database.





# Caché Accessibility

- Choice of tools, languages, and modes for accessing Caché data.
- Applications can use combination of object, relational, and multidimensional access.





# Cache History

**Neil Pappalardo** develops **MUMPS**  
(*Massachusetts General Hospital Utility Multi-Programming System*)

MUMPS Development Committee (MDC) develops **ANSI standard**

Many vendors  
2nd and 3rd ANSI standard  
MUMPS renamed to **M**

InterSystem purchases other M-vendors

InterSystems  
launches new Product: **Cache**

1967

1977

1980 ... 1990

1993

1997



# Working with Caché

---

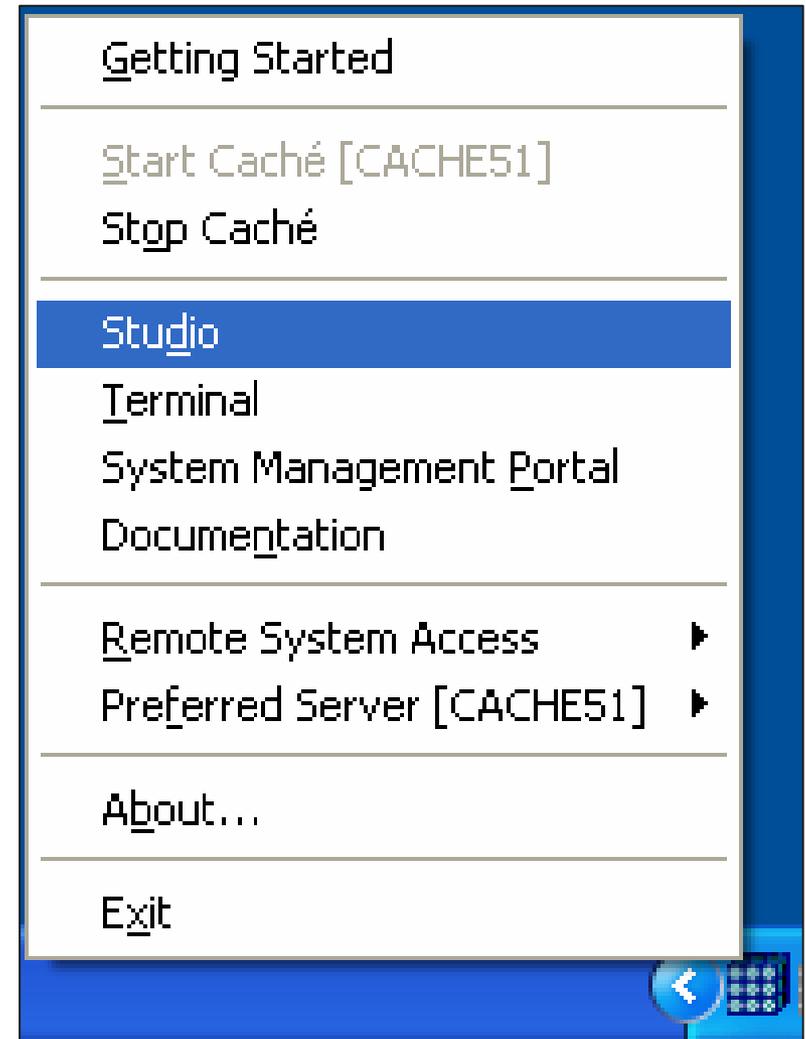
- Installation 
- Starting Caché
- Development Setup
- Connection to Remote Caché Server Neptun
- Management Portal



# Caché Cube



- Start Caché:
  - Start > Programs > CACHE > Start Caché
- Cube icon appears in Windows system tray.
  - Click cube for menu.
- *Start and Stop Caché* on local system only.
- Studio, Terminal, System Management Portal, and Documentation run on *Preferred Server*.





# Interactive Tools

---

- Studio.
  - integrated development environment (IDE)
  - not necessary for Jalapeño
- Terminal.
  - command line interface.
- System Management Portal.
  - Browser-based tools for system managers, operators, and developers.
- Remote System Access.
  - allows one system to run GUI components on remote servers.



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Development Setup

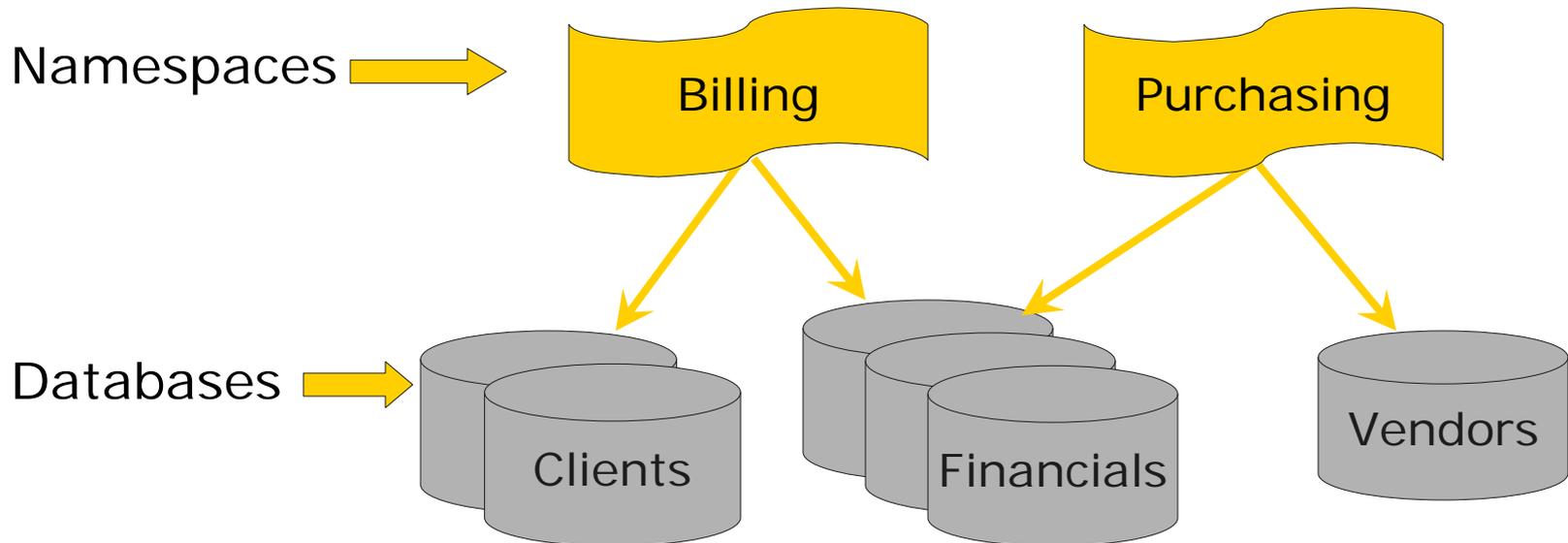
---



# What is a Namespace?

## Basic Structure

- Data and code are stored in Caché in *databases*, referred to by *namespaces*.
  - Database is a physical storage location.
  - Namespace defines logical references to several databases.

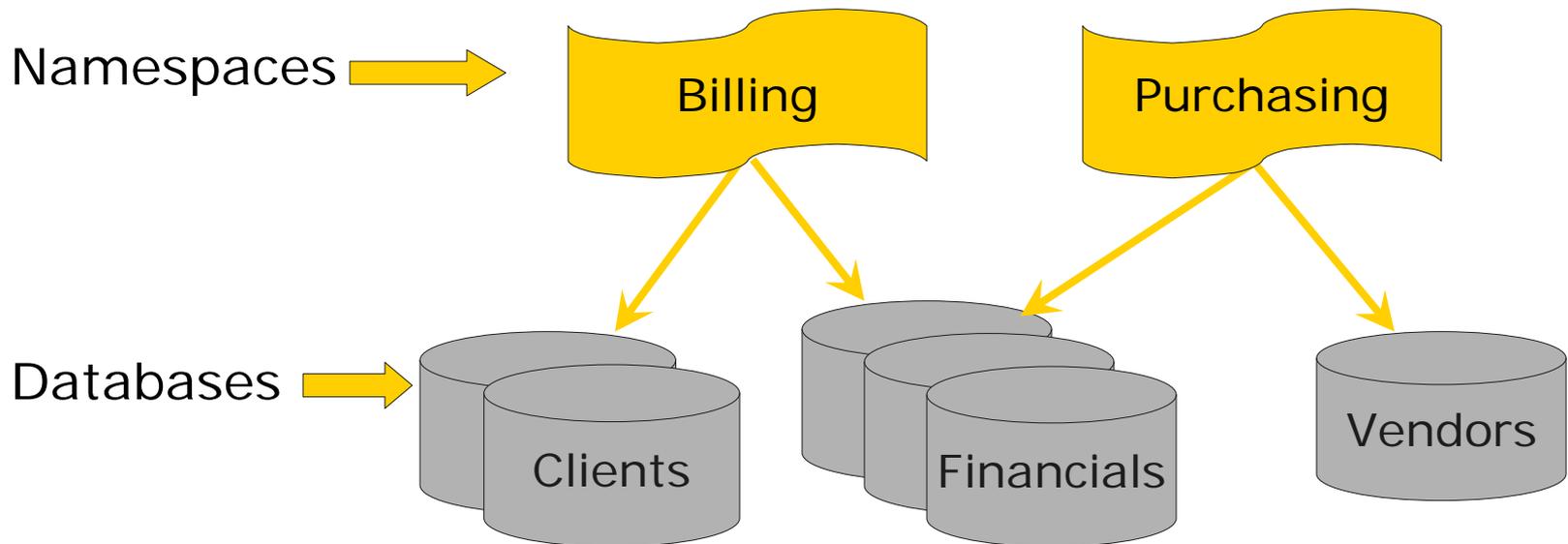




# What is a Namespace?

## Basic Structure (cont.)

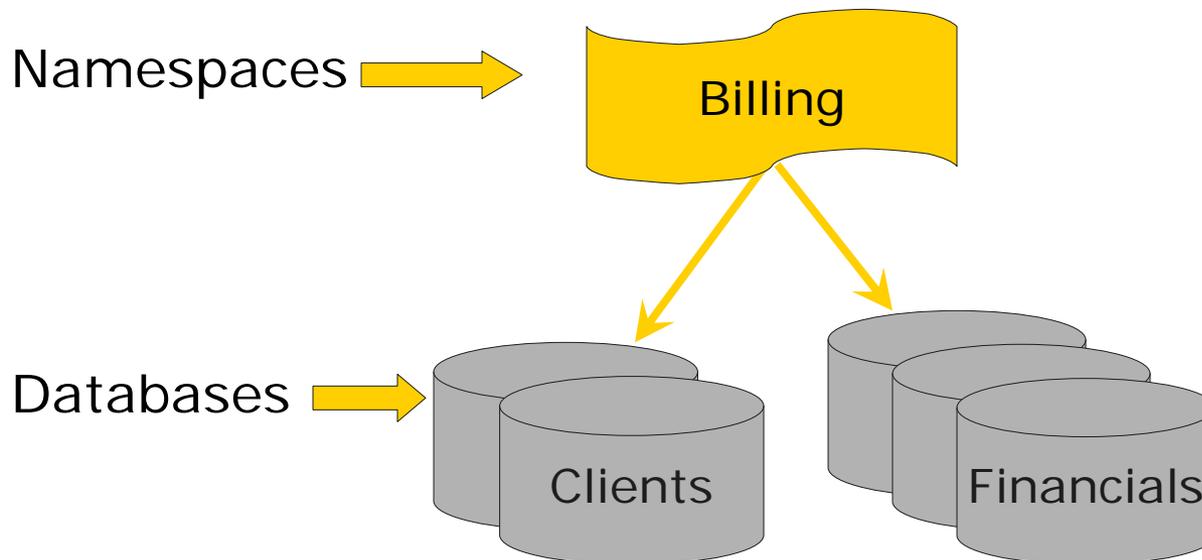
- Each namespace has default (primary) database.
- Several namespaces can refer to same database.





# Namespace Benefits

- Allows code in one database to access code and data in other databases, regardless of physical location.
- Allows code and data to be distributed among multiple databases.





# How To: Pick Namespace (cont.)

- In Portal, click namespace on left.

INTERSYSTEMS Classes  
Licensed to: License missing or unreadable.

Server: SOLONVIR  
Instance: CACHE51  
User: Administrator

Home | About | Help | Logout  
[Home] > [Classes]

NAMESPACES  
DATABASES

%SYS  
DOCBOOK

SAMPLES  
USER

View Classes | View Routines | View Globals | Compile | Export | Import | Delete

Classes in namespace SAMPLES: Last update: 2005-09-22 13:47:29.961  Auto

Classes: \*   System Filter: Page size: 100  Items found: 46

| Name                     | Size | Date                |                               |
|--------------------------|------|---------------------|-------------------------------|
| BasTutorial.Person.cls   | 311  | 2005-08-18 02:07:17 | <a href="#">Documentation</a> |
| Cinema.Duration.cls      | 398  | 2005-08-18 02:07:17 | <a href="#">Documentation</a> |
| Cinema.Film.cls          | 1187 | 2005-08-18 02:07:17 | <a href="#">Documentation</a> |
| Cinema.FilmCategory.cls  | 496  | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.RemoveCookie.cls  | 1628 | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.Show.cls          | 548  | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.Theater.cls       | 333  | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.TicketConfirm.cls | 1345 | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.TicketItem.cls    | 699  | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.TicketOrder.cls   | 435  | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |
| Cinema.Utils.cls         | 3323 | 2005-08-18 02:07:18 | <a href="#">Documentation</a> |



# Databases

- Each Caché database represents CACHE.DAT file in OS file system.
- Database contains *routines* (code) and *globals* (data).
- Data and code may be spread among multiple databases.

| Block Format | Maximum Size | Support                 |
|--------------|--------------|-------------------------|
| 2 KB         | 16 GB        | Upgrade-only as of v5.1 |
| 8 KB         | 32 TB        | v4.1+                   |



# Installation Databases

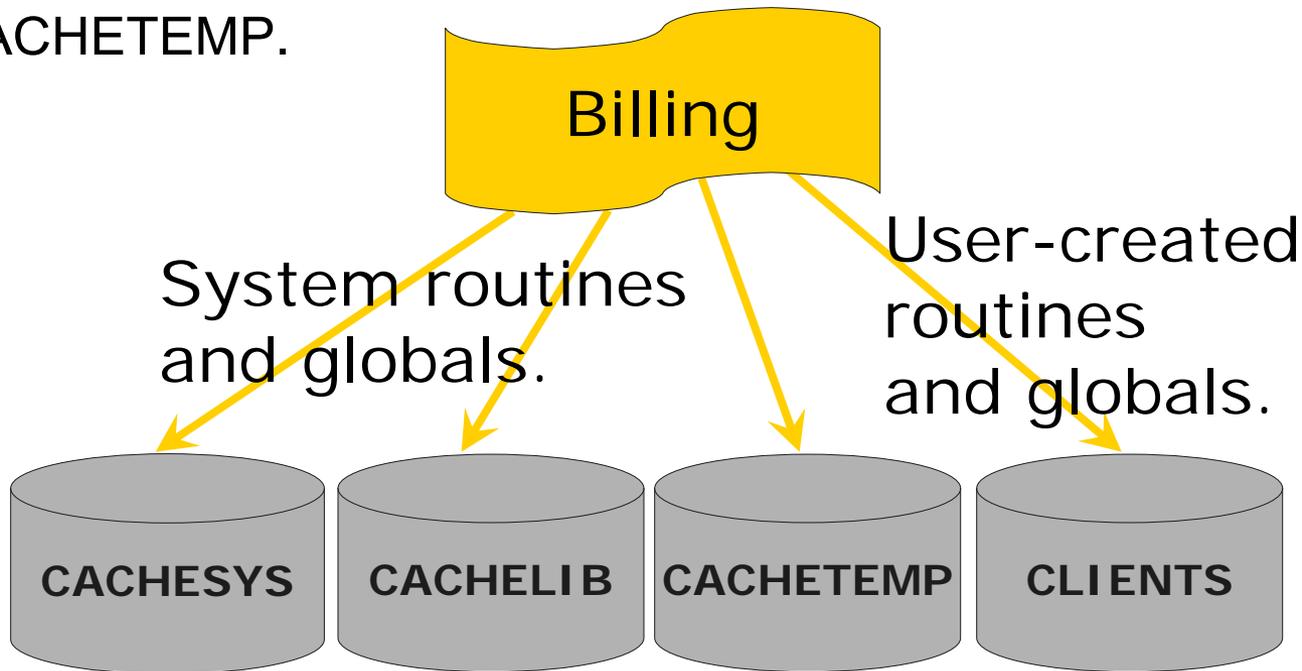
---

- System databases:
  - CACHESYS (System manager database).
  - CACHELIB (System-wide utilities).
  - CACHEAUDIT (Event logging).
  - CACHETEMP (Temporary storage).
  - DOCBOOK (Documentation).
- Testing/learning databases:
  - SAMPLES (Samples).
  - USER (Empty).



# Namespace Default Mappings

- By default, each namespace maps to:
  - CACHESYS.
  - CACHELIB.
  - CACHETEMP.





# USER and SAMPLES

---

- Caché provides two general purpose namespaces, for simple testing, development, and learning:
  - USER is empty, and not affected by a Caché upgrade.
  - SAMPLES contains sample classes, code, and data, and is completely restored by a Caché upgrade.
- For development of a large application use other namespaces.



# Development Sandbox

---

- Use DAPROnn accounts, namespaces and databases on the Neptun-Server, or
- On your own system or in lab T013 create a “sandbox” for development work, consisting of:
  - Namespace.
  - Database.
  - *Resource and Role.*
    - Control access to database.
- Create each one individually, or start by creating a new namespace.
  - Prompts to create new database.
    - Prompts to create new resource, and role with read/write access on resource.



## Development Sandbox (cont.)

---

- Creator must be a member of %Manager role.
  - Installer of Caché is a member of %All role, which encompasses %Manager.
- In lab T013 create new namespace on drive D:  
e. g. on D:\CacheDB\
- After shutting down the database save  
CACHE.DAT file on your home directory.



# How To: Create a Namespace



- Using Portal, click *Configuration* → *Namespaces* → *Create New Namespace*.
- Specify name.
- Choose an existing database, or click *Create a New Database*.
- To allow CSP applications (web applications with Caché Server Pages) access to namespace, click *Create a default CSP application....*
- Click *Save*.



# How To: Create a Database (cont.)



- Assign resource, by clicking one of:
  - *%DB\_%DEFAULT*: catch-all for databases without database-specific resource.
  - *Use an existing resource* (pick from list).
  - *Create a new resource*.
- Click *Next* to confirm choices. Click *Finish*.



# How To: Create a Database Resource



- Using Portal, click *Security Management* → *Resources* → *Create New Resource*.
  - Or, start Wizard when creating database.
- Specify name (*%DB\_Database*) and description.
- Don't click *This is a public resource*, in order to restrict access using associated *%DB\_Database* role.
  - Or, allow it to be a public resource, and assign *Read* or *Write* permissions.



# How To: Create a Database Resource (cont.)



- Click *Save*. Resource and role created.
- Click *Security Management* → *Roles*.
- Verify `%DB_Database` role has Read/Write privileges on `%DB_Database` resource.



# How To: Switch a User to a New Default Namespace



- Using Portal, click *Security Management* → *Users*.
- Find the user, and click *Edit*.
- Change *Default Namespace* to the desired namespace.



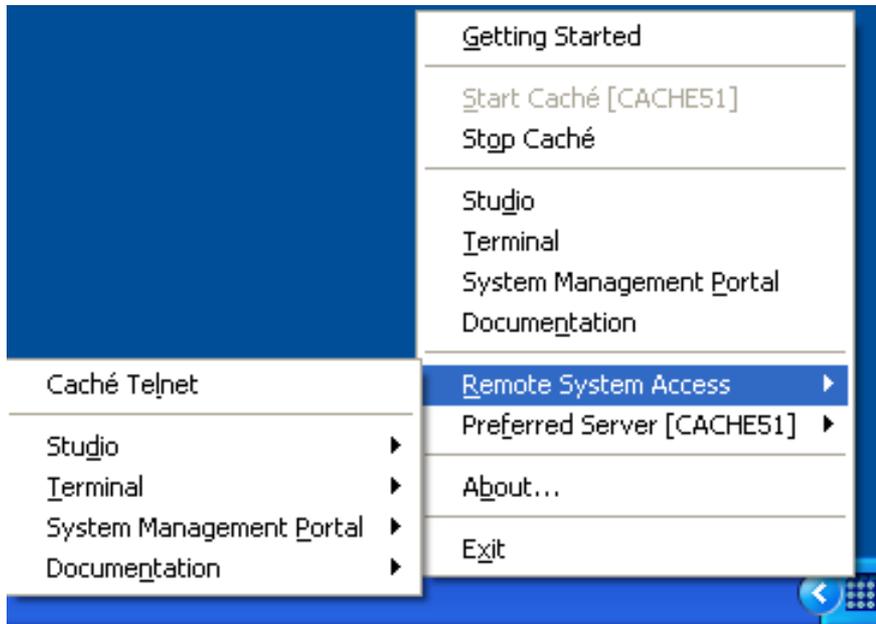
# How To: Open Access to Class Documentation



- Using Portal, click *Security Management* → *CSP Applications*.
- Find the */csp/documatic* application, and click *Edit*.
- Remove *%Development* from *Resource required to run the application*. Click *Save*.



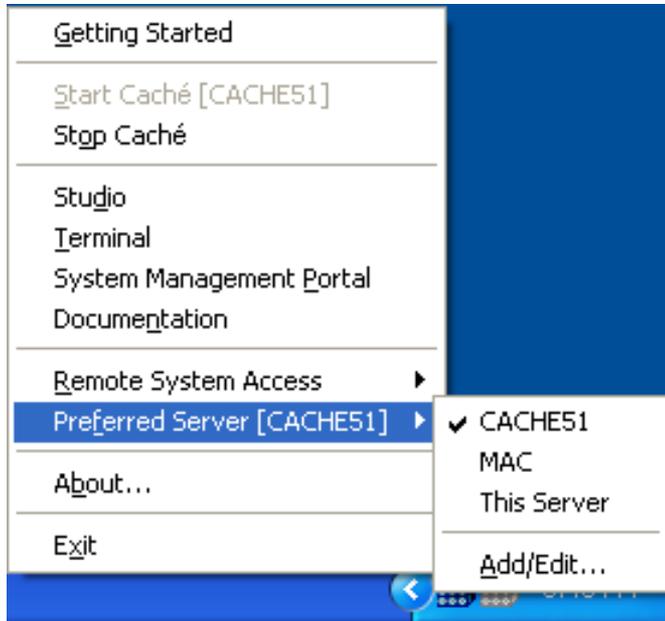
# How To: Use Remote System Access



- Click a component.
- Click name of remote server on which to run component.



# How To: Define Preferred Server



- Click Add/Edit to define list of remote servers. Specify:
  - *Name.*
  - *IP Address.*
  - *Caché SuperServer port.*
- Designate one as *Preferred Server.*



# Connection to Remote Caché Server Neptun



- From outside of HRW use SSL-Connection, e.g. the program **putty.exe**
- Map local ports to remote ports:
  - 1972 → 11972 , 23 → 10023, 57772 → 15772
  - Remote Server: neptun.fbe.fh-weingarten.de
  - Connection Server: uranus2.hs-weingarten.de
- Cube: Favorite Server > Add/Update
  - Add a new entry: puttyNeptun, neptun.fbe.fh-weingarten.de, 11972, 10023, 18972
  - Choose puttyNeptun as favorite Server



# Connection to Windows Terminal Server

---

- If you don't want to install even the client components of Caché on your PC **connect** to the **Windows Terminal Server (WTS)**
- Redirect a local port to the WTS using **putty.exe**
  - `C:/programme/putty -L 10111:141.69.55.133:3389 <yourAccount>@uranus2.hs-weingarten.de`
- Start Terminal Emulation
  - Start `mstsc.exe`
  - Computer: `localhost:10111`
- Logon to Windows with your account
- Use Caché on WTS as if you were in room K102



# Documentation

---

- Caché documentation stored in Caché database.
  - Stored/retrieved as XML.
- Fully searchable, by document element.
- Displayed as web pages.
- On Cube: Right click → Documentation 



# SQL Components

---

- Caché SQL Server
  - Allows connection from any ODBC/JDBC-compliant application or development tool.
- Caché SQL Gateway
  - Provides connection to external ODBC-compliant databases.
  - Provides relational and object access to data.



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

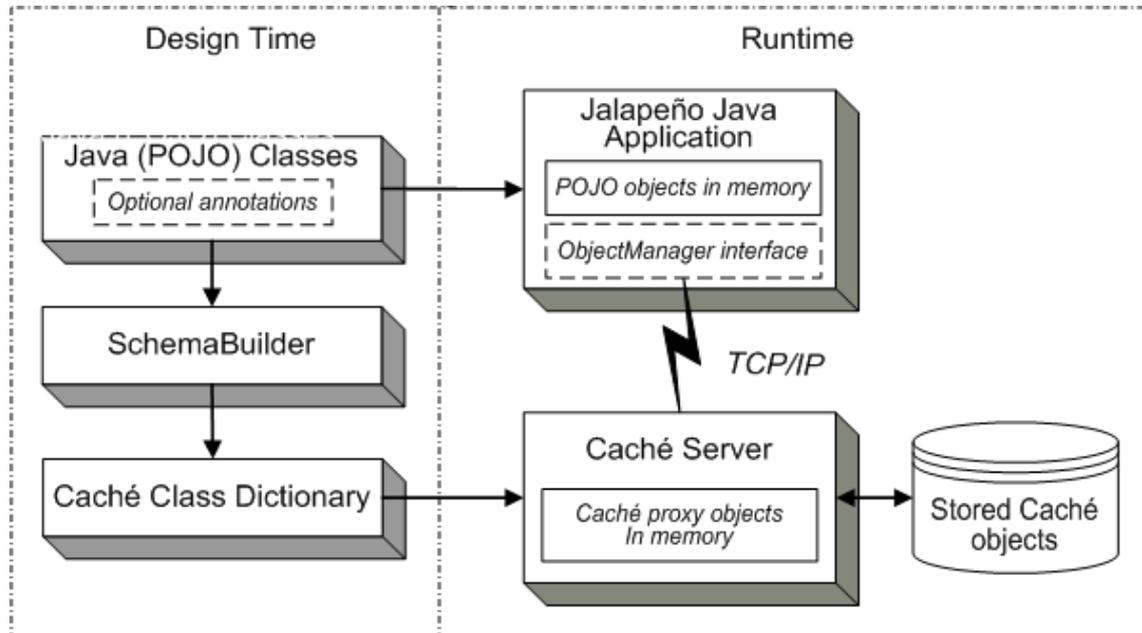
# Development of Applications

---

Jalapeño

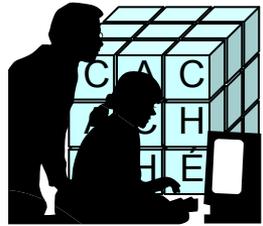


# Jalapeño Architecture





# Working with Caché Jalapeño



- Write a JAVA program.
- Define classes for persistent objects as pojos (plain old java objects)
- Import `com.jalapeno.ObjectManager`
- Use methods of the `ObjectManager` class to insert, update, delete or find objects in the database.
- Integrated development environment (IDE)
  - recommended: Eclipse
  - use the Jalapeño plugin
  - [Description](#) for installing Jalapeño plugin



# Demonstration: Create Persistent Room Class



- In Eclipse create Room class
- Connect to Database
- Mark Room class as persistent
- Create DB-Schema for Room class
- Write main method
  - connect to database
  - create a Room object
  - insert Room into database
- View database in Management Portal



# Unit Testing

---

- *Unit testing*, one of the 12 practices of Extreme Programming, is a formalized way to test a module of code, and prove that it works.
  - Module could be a class, or a method only.
- Unit tests are a sort of specification of the behaviour of your classes.



## Unit Testing (cont.)

---

- Exercises use *test driven development* model.  
You will:
  - Use pre-written unit testing methods for each module you're about to create.
    - You'll write your own unit tests later.
  - Run tests **before** creating module, and see them fail.
  - Create module, and re-run tests, fixing problems in module until all tests pass.



# Unit Testing Benefits

---

- Proving a module works.
  - Proving you've correctly followed instructions.
- Regression testing when modules change.
- Focusing your mind on goal/usage of a module before developing it. Each step of a test is:
  - Run some lines of code that use module.
  - Assert what you think result should be.



# Unit Testing Framework

---

- Caché provides a Unit Testing framework, similar to other xUnit frameworks for Object Script
  - %UnitTest.TestCase class for defining test cases.
  - %UnitTest.Manager class for running test cases.
  - Web page for viewing test results.
- For Jalapeño use JUnit Tests
  - Actual version: JUnit 4
  - Integrate JUnit 4 into Eclipse



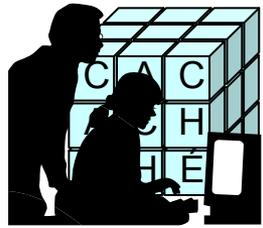
# Create Unit Tests for Room Class: Specification for Room

---

- Every room has a unique room number
- A room has a type (optional) like "office" or "lab"
- The size of a room (optional) is a positive number
- A room may have one or more installed **devices**
  - A device has a manufacturer (optional)
  - A device has a serial number (optional)
  - A device has a type (optional)
  - Every device belongs to exactly one room



# Create Unit Tests for Room Class: Demonstration Test Development



- Create JUnit tests for Room class
  - Run tests: tests fail
- Alter Room class to meet the specification
  - Run tests again: tests succeed
- Add tests for Device class
  - Run tests: tests fail
- Create Device class according to specification
  - Run tests: tests succeed



## Exercise



- Add additional specification for Device class
- Enhance test cases for Device class
  - Run tests: tests fail
- Alter Device class
  - Run tests again: tests succeed



# Objects in Memory and Database

---

- `new <constructor>`  
creates new object in memory
- `objectManager.insert(<object>, <deep>)`  
stores object in database
- `objectManager.openById(<class>, <id>)`  
retrieves an object from database into memory.
- `objectManager.update(<object>, <deep>)`  
stores altered object in database
- Remove an object from memory: `<object> = null`
- `objectManager.removeFromDatabase(<object>)`  
deletes an object from database



# Objects

- A persistent class contains its objects along with their data.
  - The *extent* of a class is the collection of all its objects.

pojos.Room class



T111, lab, 100



L022, office, 16



N042, lecture hall, 300



# Tables

- Persistent objects are rows in relational tables.
  - Object id number is ID column.
  - Package name is schema name.
    - *User* package is *SQLUser* schema.
- `pojos.Room` table:
  - Contains extent of `pojos.Room` class.

| ID | RNr  | Type         | Size |
|----|------|--------------|------|
| 1  | T117 | lecture hall | 100  |
| 2  | K102 | lab          | 80   |
| 3  | T011 | office       | 16   |



# System Management Portal

---

- System Management Portal is administration front end for Caché.
- Caché must be running to use Portal.
- Open several Portals to avoid jumping from function to function.



System Management Portal



# Portal Layout



- Portal organized in three sections, based on roles.
  - System Administration.
    - Restricted to certain role(s).
  - Data Management
    - Accessible by most roles.
  - Operations.
    - Restricted to certain role(s).
- Eight major groupings of tasks.
- Use *Go To* to jump directly to common tasks.



# Architectural Configuration

---

- Settings for architecture of system.
- In *System Administration* section, click *Configuration*.
- *System Configuration* section.
- Examples:
  - Namespaces and databases.
  - Memory allocation.
  - Startup (port selection).



# Operations Configuration

---

- Settings that govern operations.
- In *System Administration* section, click *Configuration*.
- *System Configuration* section.
- Examples:
  - Backups.
  - Journals.
  - Task Manager.



# Operations

---

- Operational tasks.
- *Operations* section.
- Examples:
  - Running backups.
  - Reviewing running processes.
  - Reviewing *System Dashboard* (v5.1).



## Exercise



Description: View the list of running processes on Neptun.

Instructions:

- Using the Portal, in the *Operations* section, click *Processes*.
- Find your process(es) by looking for *DAPRO* in the *User* column.
- Click on the *User* column to sort the list by this value.
- Enter *DAPRO* in the *Filter* field to filter list.



# Classes, Routines, Globals

---

- Application component management tasks.
- In *Data Management* section, click *Routines*, *Globals*, or *Classes*.
- Examples:
  - Searching routines, globals, or classes.
    - Use combination of search mask and filter to produce list.
  - Compiling classes.



## Exercise



Description: View the list of classes in the `pojos` package.

Instructions:

- Using the Portal, use the *Go to* dropdown list and click *View Classes*.
- If you don't see the `pojos` classes, make sure you're in the `DAPROnn` namespace.



# SQL

---

- SQL management tasks.
- In *Data Management* section, click *SQL*.
- *Execute SQL Statement* provides access to an interactive *Query Builder*.
- Examples:
  - Reviewing table schema.
  - Viewing table data.
  - Importing data.



# Developer's Corner

- <http://www.intersystems.com/cache/devcorner/index.html>
- Latest information for developers.
- Samples and documentation.

InterSystems Caché – Education & Events

http://www.intersystems.com/cache/devcorner/index.html

Back Forward Reload Stop Location

Technology Testimonials Analyst & Media Coverage Partners Support Education & Events **Developers**

InterSystems **CACHE** INNOVATIONS

Download a better database. For free.

## Caché Developer's Corner

Caché Developer's Corner  
Caché Newsgroup  
Caché Support

Caché Main / Caché Developer's Corner

### Caché Developer's Corner

In this section, we've collected information of interest to Caché developers. The links below point to resources that will help developers [learn](#) and [use](#) Caché. There are also links that help you to interact with other members of the [Caché community](#) to help guide you in building and deploying your own Caché applications.

#### What's New

Read the [latest Release Notes](#) and [download the latest version of Caché](#).

There are many new Caché tutorials. The links below allow you to read the tutorial content.

Document: Done



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques Object Oriented Databases

---

Type Constructors,  
Complex Objects



# Requirements for OODBS

---

Object-oriented database management systems must as well match the requirements of object oriented programming as the requirements of databases.

## object-oriented perspective

- Complex objects
- Object identity
- Encapsulation
- Types and classes
- Class hierarchies
- Overloading of methods

optional:

- Multiple inheritance

## Database perspective

- Persistence
- Disk-storage organisation
- Concurrent transactions
- Recovery mechanisms
- Query languages
- Database operations

optional:

- Distributed databases
- Version administration



## Type constructors, complex objects

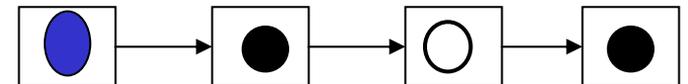
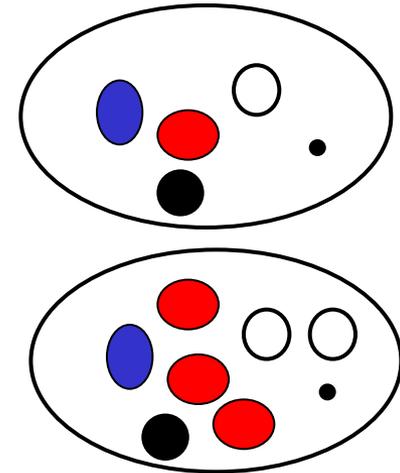
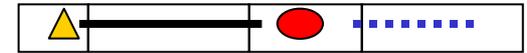
---

- To be able to store **many** objects, a **set constructor** is necessary.
- The type of a relational table is  
SET OF (TUPLE OF (Typ1 A1, . . . , Typn An))
- In OODBS user defined classes can be used besides basic types.
- TUPLE OF corresponds aggregation.
- There are different implementations of the set operator in different OODBMs



# Overview: Type Constructors

- Tuple constructor **TUPLE OF**
  - Combines several components of possibly different type
  - Corresponds to aggregation
- Set constructor **SET OF**
  - Many elements of the same type build a set.
  - Each element can only be contained once in the set.
- Multi-set constructor **BAG OF**: like SET but
  - one element can have copies in the bag
- List constructor **LIST OF**: like BAG, but
  - Sequence is of interest





## Overview: Type Constructors (2)

- Array constructor ARRAY OF: like LIST, but

- may contain gaps
- index of type long integer

|         |       |   |        |     |
|---------|-------|---|--------|-----|
| 0       | 1     | 2 | 3      | ... |
| Atlanta | Paris |   | London |     |

- Dictionary constructor  
DICTIONARY OF (key, value)

- unordered sequence of (key, value)-pairs
- lookup operation  
returns *value* to a given *key*

| Key | Value        |
|-----|--------------|
| A   | excellent    |
| B   | very good    |
| C   | good         |
| D   | satisfactory |
| E   | sufficient   |
| F   | fail         |

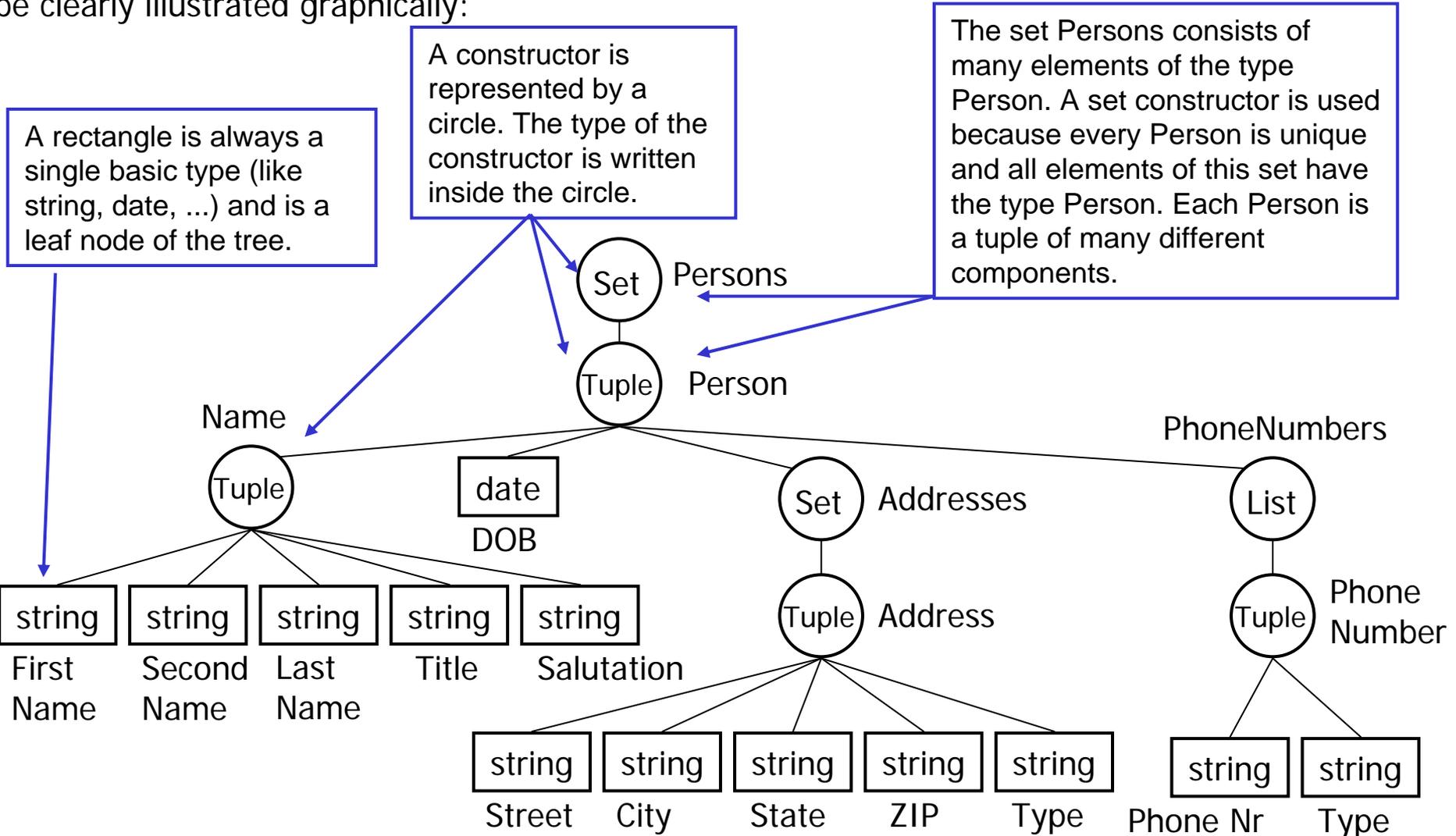
- Type constructors are used recursively
- Each type constructor has some typical operations. These are discussed later (see "Operations on Complex Types").



# Example: Type Construction for Persons

## Graphical Representation

Constructors can be used recursively, which allows a tree structure for complex types. This can be clearly illustrated graphically:





## Example Type Construction in O<sub>2</sub>: Persons

---

SET OF

TUPLE OF (

  Name: TUPLE OF (

    FirstName: STRING, SecondName: STRING

    LastName: STRING, Title: String ...),

  Adresses: SET OF TUPLE OF (

    Street: STRING, City: STRING,

    ZIP: STRING, State: STRING, Type: String),

  PhoneNumbers: LIST OF TUPLE OF (

    PhoneNr: STRING, Type: STRING),

  DOB: DATE)



# ODMG Standard

- ODL (object definition language) for ODMG-types
  - extension of IDL (interface definition language)
  - similar to C++ but language independent
- All classes inherit from interface **Object** that provides
  - constructor function (interface ObjectFactory)
  - locking information and operation
  - identity comparison function **same\_as(object)**
  - copy function
  - delete function
- Different constructors for collection types

Literature:

R.G.G. Cattell. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, San Francisco, California, 1997.

F. Matthes, J.W. Schmidt: Vorlesung "Datenbanken und Informationssysteme" WS 98/99,

<http://www.sts.tu-harburg.de/teaching/ws-98.99/DBIS/52-dbis.pdf>, visited 27-Jun-2006



## ODMG Standard (cont.)

- Collections supported by the ODMG-Model
  - Set <t>
  - Bag <t>
  - List <t>
  - Array <t>
  - Dictionary <t, v>
  - where t and v are any types
- Collections supported for objects and for literals
- Atomic types
  - long, long long, short ...
  - float, double
  - boolean, octet
  - char, string
  - enum
    - type generator
    - can only take values listed in declaration
- Structured literals
  - date
  - time, timestamp
  - interval
  - user defined with **struct**



## Example: Persons in ODMG-Model

```
Class Person (extent Persons)
{attribute NameClass Name;
 attribute date DOB;
 attribute set <Address> Addresses;
 attribute list <PhoneNr> PhoneNrs; }
```

set of objects

```
Class NameClass
{attribute string FirstName;
 attribute string SecondName;
 attribute string LastName;
 attribute string Title;
 attribute string Salutation; }
```

no own set  
of objects



## Example: Persons in ODMG-Model

---

**Class Address**

```
{attribute string Country;
 attribute string Ci ty;
 attribute string Street;
 attribute string ZIP;
 attribute string State;
 attribute string Type; }
```

**Class PhoneNr**

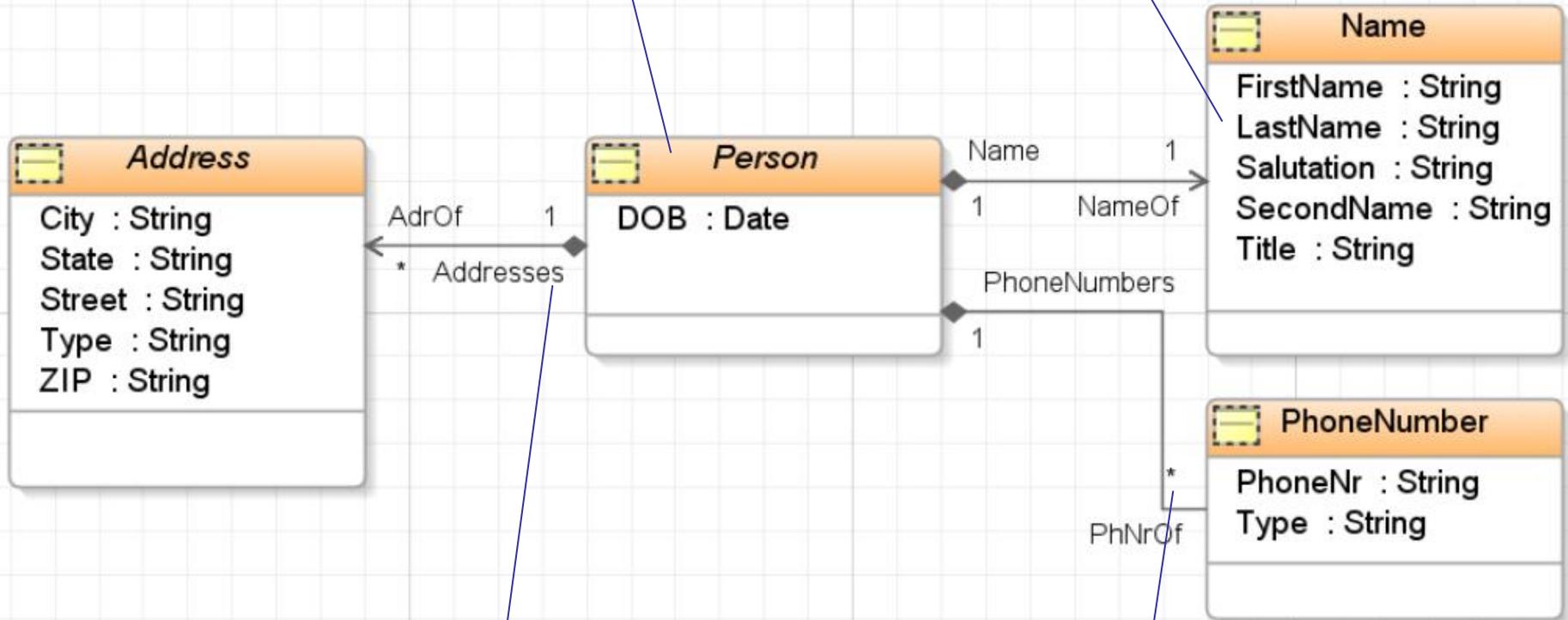
```
{attribute string PhoneNr;
 attribute string Type; }
```



# Type Construction with UML

Set Constructor

Tuple Constructor



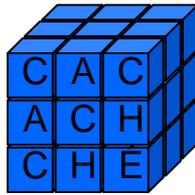
complex properties

List or Array Constructor

No dictionary constructor



# Type Constructors in Caché Jalapeño

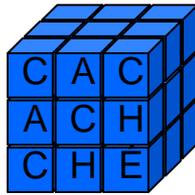


Caché classes are defined by JAVA code if Jalapeño is used.

- **Tuple:** Attributes of a class
- **Set:**
  - A Class automatically gets an **extent** if it is marked as **persistent** using the menu commands of Eclipse:  
Jalapeno > Schema Management > Mark as persistent
  - **No** set-operator for components:  
`private Set <ElementType> setAttribute;` **not possible**
  - Use list- or array-constructor instead, or relations
- **Bag:** not supported
- **List:**
  - include `java.util.List` and `java.util.ArrayList`
  - Define a list attribute in the class like  
`private List <ElementType> listAttribute;`



# Type Constructors in Caché



- **Array:** not directly supported, use dictionary with integer key values
- **Dictionary:**
  - include `java.util.Map`
  - Define a map attribute in the class like  
`private Map <KeyType, ElementType> mapAttribute;`
  - This will be translated to a Caché dictionary  
`Property mapAttribute As array Of ElementType;`
- Array in Caché is Dictionary in ODMG-Model (Map in Java)





# Example: Persons in Caché Jalapeño



```
public class Person {
 private NameClass name;
 private Date dob;
 private List <Address> addresses;
 private List <PhoneNumber> phoneNumbers;
 public NameClass getName () { return name };
 ... // get- and set methods
 public String toString {...}
}
```

```
@Embeddable
public class NameClass {
 private String firstName;
 private String secondName;
 private String lastName;
 private String title;
 private String salutation;
 // get- and set-methods
}
```

no own set of objects,  
Names are only  
component objects in  
Person class



## Example: Persons in Caché Jalapeño



```
@Embeddable
public class Address {
 private String city;
 private String street;
 private String zip;
 private String country;
 private String state;
 private String type;
 // get- and set-methods
}

@Embeddable
public class PhoneNumber {
 private String phoneNr;
 private String type;
 // get- and set-methods
}
```



## Type Construction: Cars

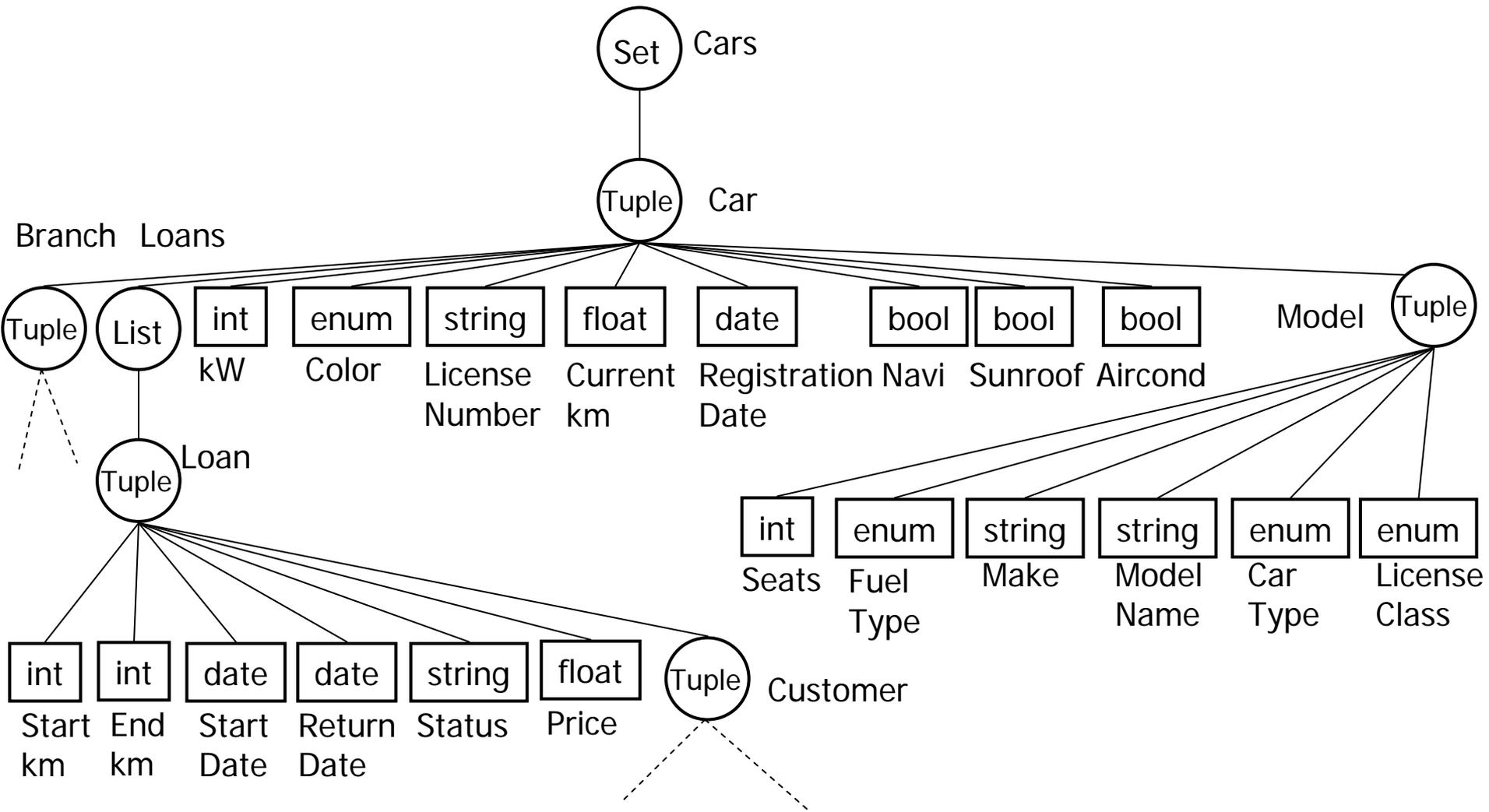


- Practice: Define the object type of Cars as complex type graphically:
  - There may be many cars of one model, e. g. Opel Astra TDI 2.0.
  - Car type or class is a value like "Limousine", "Cabrio" or "SUV"
  - A car has a list of Loans.
- What are the consequences of recursive types, e.g. if Person contains a set Friends of type Person?



# Type Construction for Cars

## Solution: Graphical Representation





# Operations on Complex Types

---

- Tuple constructor
  - Access components of a tuple
  - Test two tuples on equality
  
- All collection operators
  - Test if collection is empty
  - Test if collection contains a certain element
  - Access to all elements (iterator)
  - Insert, update, or delete element



## Operations on Complex Types (cont.)

---

- Set constructor
  - Set comparisons: = (equality) ,  $\subset$  (subset)
  - Set operations:  $\cap$  (intersection),  $\cup$  (union),  $\setminus$  difference
- Bag constructor
  - Number of occurrences of an element
  - Set operations: intersection, union, difference
  - No set comparisons
- List constructor
  - Access in sequence of list
  - Insert at certain position or after certain element
  - Concatenation of lists



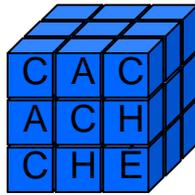
## Operations on Complex Types (cont.)

---

- Array constructor
  - Access to elements at certain index
  
- Dictionary constructor
  - Access to elements at certain key
  - Test whether certain key is in dictionary
  - Bind a value to a certain key (insert)
  - Unbind a value from a key (delete)



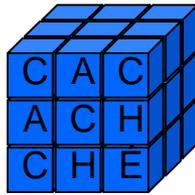
# Operations on Complex Types in Caché Jalapeño: Object Access



- Access to components by setter- and getter-methods  
`p1.getName().getFirstName()` if `p1` is a `Person` object
- Test if object exists in database:  
`objManager.getId(p1) != null`
- Retrieve stored objects into memory: `Person p1 = (Person) objManager.openById(Person.class, myId);`
- Insert an object: `objManager.insert(p1, true);`
- Update an object: `objManager.update(p1, true);`
- Insert or update an element: `objManager.save(p1, true);`
- Delete an object: `objManager.removeFromDatabase(p1);`  
`objManager.removeFromDatabase(Person.class, myId);`
- Iterator to access all objects of a class: `Iterator persList = objManager.openByQuery(Person.class, null, null);`



# Methods for List- and Array- (Dictionary-) Collections

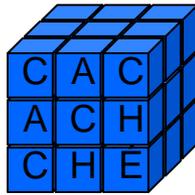


For lists use the interface *List* and the implementation *ArrayList* of `java.util`  
For dictionaries use the interface *Map* and the implementation *HashMap* of `java.util`

| Method                                                                                                        | Purpose                                                            | Success                | Failure         |
|---------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|------------------------|-----------------|
| List: <code>add(object)</code><br><code>add(index, object)</code><br>Map: <code>put(key, value)</code>        | Add <i>object</i> to collection                                    | true<br>-<br>old value | false<br>-<br>- |
| List: <code>addAll(list)</code>                                                                               | Appends <i>list</i>                                                | true                   | false           |
| List: <code>contains(value)</code><br>Map: <code>containsKey(key)</code><br><code>containsValue(value)</code> | Tests whether <i>value</i> or <i>key</i> is in collection          | true or false          |                 |
| List: <code>get(index)</code><br>Map: <code>get(key)</code>                                                   | Get <i>value</i> at <i>index</i><br>Get <i>value</i> at <i>key</i> | <i>value</i>           |                 |



# Methods for List- and Array- (Dictionary-) Collections (cont.)



| Method                                                                                      | Purpose                                                                            | Success                                                       | Failure |
|---------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|---------------------------------------------------------------|---------|
| List: remove ( <i>index</i> )<br><br>remove ( <i>object</i> )<br>Map: remove ( <i>key</i> ) | Remove object at <i>index</i><br><br>Remove one <i>object</i><br>Remove <i>key</i> | <i>value</i> at <i>key</i><br><br><i>true</i><br><i>value</i> |         |
| size()                                                                                      | Count items                                                                        | integer                                                       |         |
| clear()                                                                                     | Empty collection                                                                   |                                                               |         |



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques

## Object Oriented Databases

---

### Classes and Relations



## Classes: Type- and Set-View

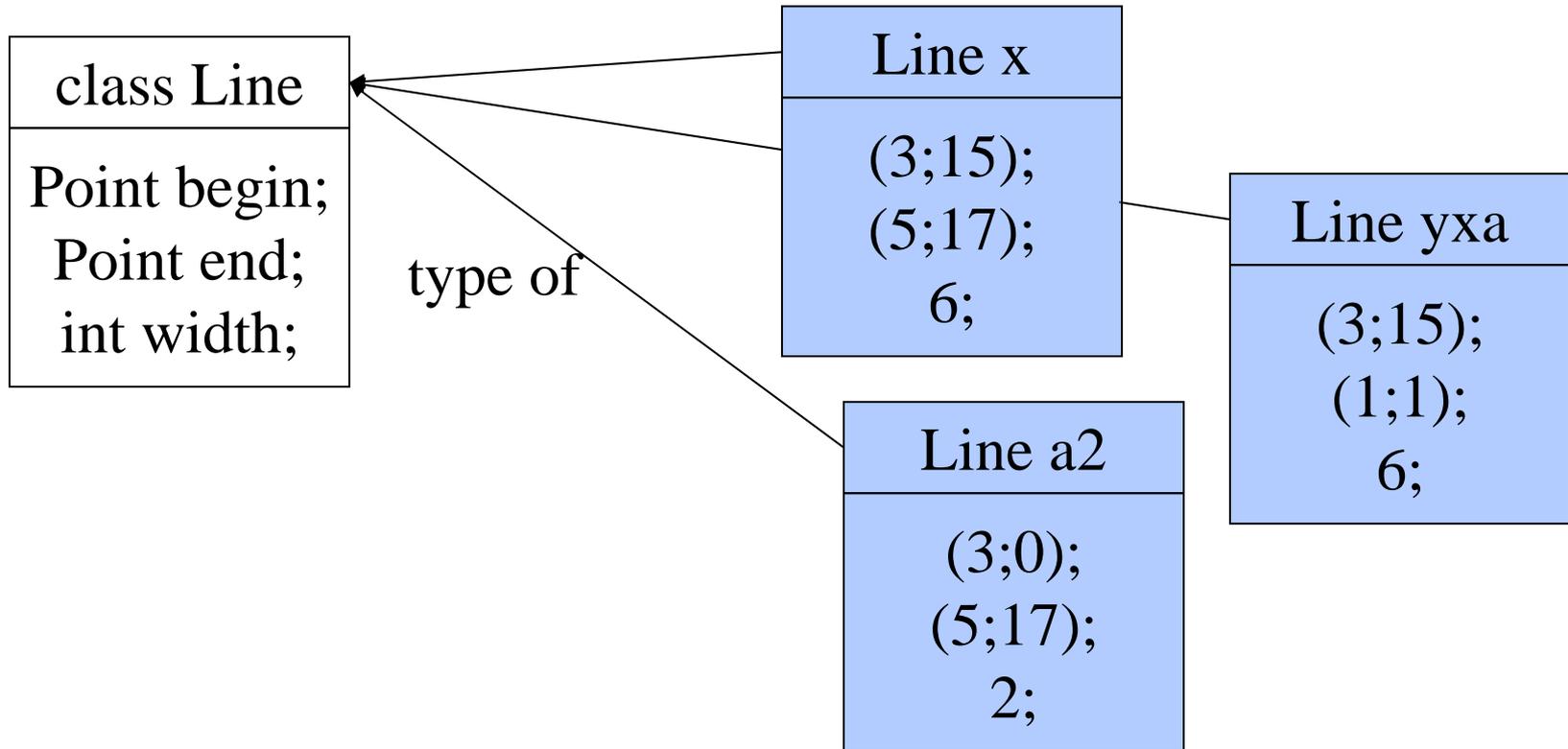
---

The notion **Class** has different meanings:

- Set of objects belonging together
- Container for all hitherto created objects of the class
- Type = construction schema of the objects
- Domain for (abstract) objects



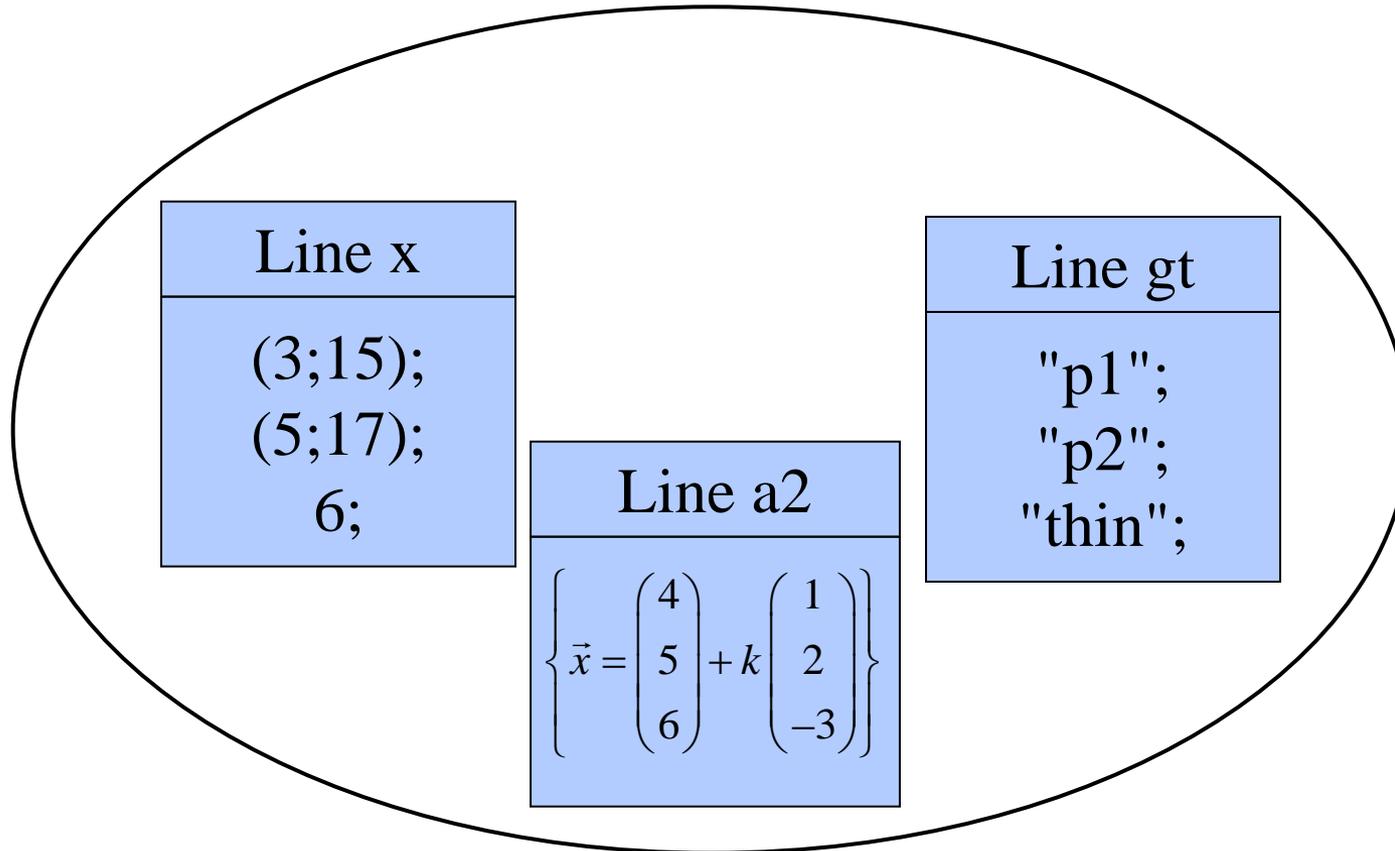
# Concepts for Classes: Type-based



- A class defines a complex type
- Objects of the class are not gathered in a container
- This is the situation in programming languages like C++



## Concepts for Classes: Set-based

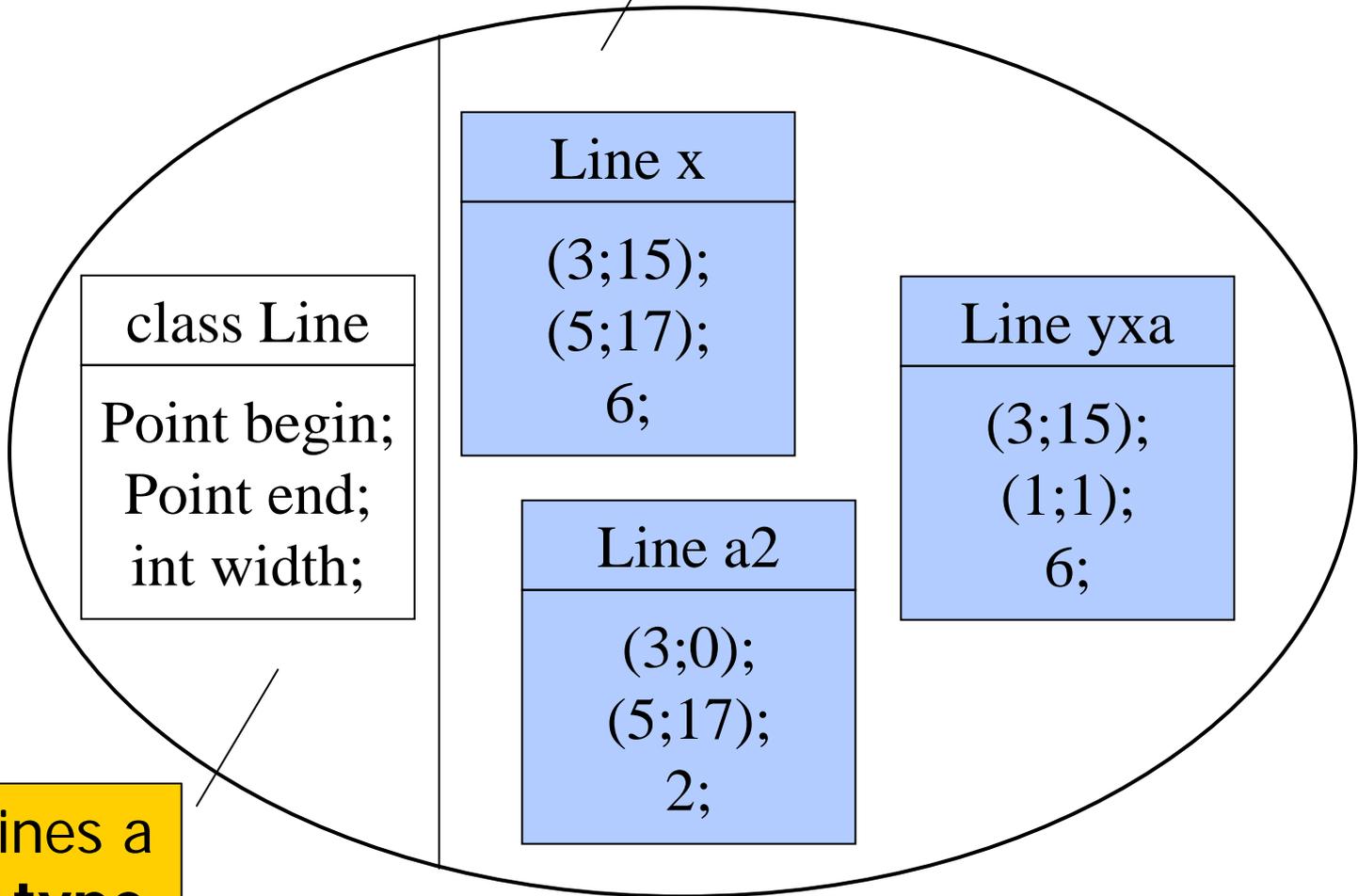


- A class is a container
- Types of the elements are not fixed



# Set-type-based

**AND** class is **container** for objects (extent)



class defines a complex **type**

This is the solution in OODBMSs



## Relations between Classes

---

- Every class consists of attributes  
(= properties = components)
- Their types can be atomic types or classes.
- Class-componentclass-relations implement relationships between different classes
- Distinguish: class-subclass-relation → see module "Inheritance"
- Different semantics of component classes
  - shared/private component objects
  - dependent/independent component objects
  - encapsulated/standalone component objects
  - unidirectional/mutual component relation



# Shared/Private Component Objects (Cardinality of Relationships)

---

- Shared component objects:
  - A component object is component of many objects
  - An object can be component of objects of different classes
  - M:N(1)-relation between component class and class
  - Example: *Task* is shared component of *Employee*
- Private component objects:
  - Each component object is component of only one object
  - 1:N(1)-relation between class and component class
  - Example: *Vehicle* is private component of *Branch*



# Dependent/Independent Component Objects

---

- Dependent component objects ...
  - ... only exist together with their master object
  - ... are created/deleted together with their master object
  - Shared dependent objects are deleted with last object they are component of
  - Example: *Name* is dependent of *Person*
- Independent component objects ...
  - ... can exist without another object
  - ... are created and deleted independently
  - If you delete a component object, take care of objects which have it as a component.
  - Example: A *Branch* is an independent component of a *Vehicle*



# Encapsulated/Standalone Component Objects

---

- Encapsulated component objects ...
  - ... are only accessible by their master objects.
  - ... are always dependent
  - Redundancy when used for M:N-relations
  - Example: *Name* is encapsulated in *Person*
- Standalone component objects ...
  - ... are accessible independently **or** by the objects which have them as component.
  - Example: *Branch* is a standalone class. A branch is accessible without a vehicle- or employee-object



# Unidirectional/Mutual Component Relation

---

- Unidirectional component relation
  - One object is obviously the superior object, the other the component
  - No direct way from the component to its superior object
  - Example: *House* → *Windows*.  
You usually do not ask: "To which house does this window belong?"
- Mutual component relation
  - Each of two objects has the other one as component.
  - Implements bidirectional relationship between two classes
  - Example: *Branch* ↔ *Vehicle*



# Different Types of Component-Classes



- Practice: Decide which is the type for the following examples of classes and component classes:

|                     | shared | privat | enc | auton | dep | ind | dir |
|---------------------|--------|--------|-----|-------|-----|-----|-----|
| Buildings – Rooms   |        |        |     |       |     |     |     |
| Countries – Towns   |        |        |     |       |     |     |     |
| Vehicles – Loans    |        |        |     |       |     |     |     |
| Lectures – Students |        |        |     |       |     |     |     |
| Books – Publishers  |        |        |     |       |     |     |     |
| Books – Pages       |        |        |     |       |     |     |     |
| Folders – Files     |        |        |     |       |     |     |     |
| Parents – Children  |        |        |     |       |     |     |     |



# Different Possibilities for Relationships:

## a) Encapsulated component class

---

### Complex objects with encapsulated objects of component class

- Suitable for für 1:1- and 1:N-relations
- Redundancy when used for M:N-Relations
- Access not symmetrical:
  - Easy from object to component
  - Difficult from component to parent object
- See example *Person* in module "Type Constructors, Complex Objects":  
NameClass is encapsulated in Person because it is marked @Embeddable



# Different Possibilities for Relationships:

## b) Mutual Component Classes

---

Two classes have each other as component

- Can be used for 1:1-, 1:N- and M:N-Relations
- Symmetrical access
- System must ensure symmetry
- In Caché Jalapeño this is guaranteed if the attributes are marked with the `@Relationship` annotation (see next slides for details).
- The relation must not have attributes of its own:  
E. g. the examination results of students in the relation "Students attend Lectures" cannot be represented
- In this case use alternative [c\) Connection Class](#)



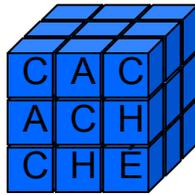
# Relationships in Caché



- A *relationship* is a two-way link between an object of one class and several objects of another (possibly the same) class.
- Relationship types:
  - Independent (One-to-many).
  - Dependent (Parent-to-children).
    - Children stored together with parent
    - However not encapsulated (children get own object identifiers)
    - Classes must be **different** in this case.
- Indirectly supported: One-to-one.
- Not supported: Many-to-many, use connection class



## Relationships in Caché (cont.)



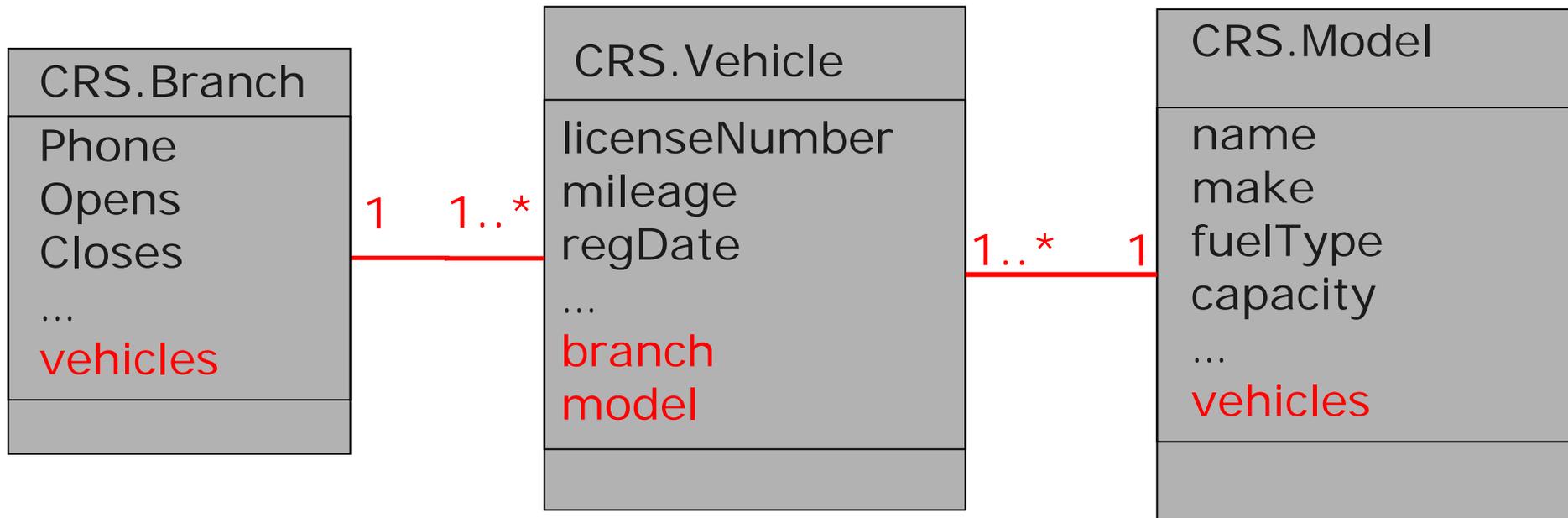
- Relationship specified in **both** class definitions
- For Jalapeño use `@Relationship` annotation.
- Example *Model – Vehicle*
  - In *Vehicle* class:

```
@Relationship(type=RelationshipType.MANY_TO_ONE,
inverseClass="Model", inverseProperty="vehicles")
private Model model;
```
  - In *Model* class:

```
@Relationship(type=RelationshipType.ONE_TO_MANY,
inverseClass="Vehicle", inverseProperty="model")
private ArrayList<Vehicle> vehicles;
```
- *One, many, parent, and children* define *cardinality* of each side of relationship.



# Example: Branch and Vehicle Class Structure

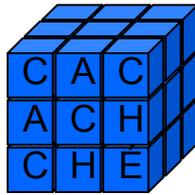


Do exercise [E-2](#)

Practice: Discuss different possibilities to implement the relation "Branch has Vehicles"



# Cardinality of Relationships in Caché



- *Cardinality*: number of elements on each side of a reference.
- **Describe** cardinality in terms of classes.
  - “CRS.Branch has a parent-to-children relationship with CRS.Vehicle. Multiple Vehicles belong to each branch.”
- **Specify** cardinality in terms of properties.
  - “The Vehicles property of the CRS.Branch class has cardinality/type *Children*. The Branch property of the CRS.Vehicle class has cardinality/type *Parent*.”



# OOP Relationship Advantages



- Relationship is a combination of a reference and a list collection.
  - Performance degrades as number of objects on many/children side surpasses 900-1000.
- Relationship provides *referential integrity* for object deletions on one/parent side.
- If you access the database using SQL a relationship is handled like a foreign key reference to a primary key.



# Referential Integrity



- For relationships between persistent classes from A to B
- One-to-Many:
  - Deletion of an object of class A fails as long as it points to at least one object of class B.
  - To delete an object of class A, delete the references to it from objects of class B.
- Parent-to-Children:
  - Deletion of an object of class A deletes all related objects of class B.
- Same is true for rows in tables.



# Using Relationships



- Link objects in two equivalent ways:
  - Insert many/children into one/parent collection.
  - Reference one/parent from many/children object.
- No control over ordering of many/children in one/parent collection.
  - To control ordering, use list- or array-constructor (see lesson "Type Constructors, Complex Objects")



## Using Relationships (cont.)



- From one/parent side, many/children side is a collection.
- To insert many/children object (e.g. *Vehicle v*) into one/parent object's collection (e.g. *Branch b*):
  - Define a setter method for the vehicles property in class *Branch*

```
public void addVehicle(Vehicle v) {
 vehicles.add(v);
}
```
  - Use this method to insert a *Vehicle v* into a *Branch b*:  
 b.addVehicle(v)



## Using Relationships (cont.)



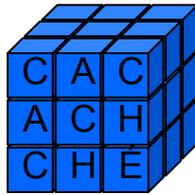
- From many/children side, one/parent side is a reference.
- To set one/parent object (e.g. *Branch b*) for a many/children object (e.g. *Vehicle v*):
  - Define a setter method for the *branch* property in *Vehicle* :

```
public void setBranch(Branch b) {
 this.branch = b;
}
```
  - Use this method to assign a *Vehicle v* to a *Branch b*:

```
v.setBranch(b);
```



# Storing Relationships



- Calling  
obj Manager. insert(x, true),  
obj Manager. update(x, true), or  
obj Manager. save(x, true)  
(i.e. with the deep parameter set to true) on either  
one/parent or many/children triggers a save operation of the  
entire relationship.
- IDs of Children in Parent-to-Children relationship depend on  
parent ID. Format: *parentID||childID*.

Do exercise **E-3!**

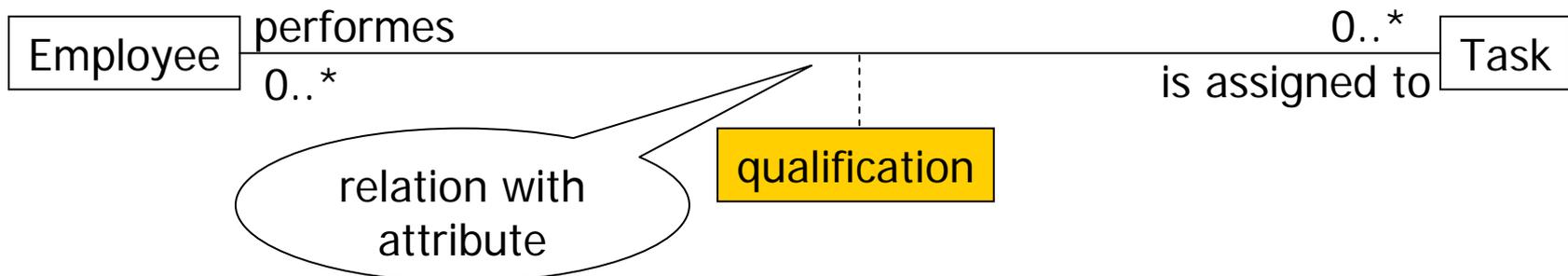
View IDs of vehicles in System-Management Portal



# Different Possibilities for Relationships:

## c) Connection Class

- The relation becomes an own class
  - Suitable for M:N-relations with attributes of their own
  - Two 1:N-relations from each class to connection class
  - In Caché necessary for every many-to-many relation
- Example: Employees - Tasks
  - The employees are assigned to different tasks, e. g. bookkeeping or car service.
  - The qualification attribute states how qualified this employee is for this task.

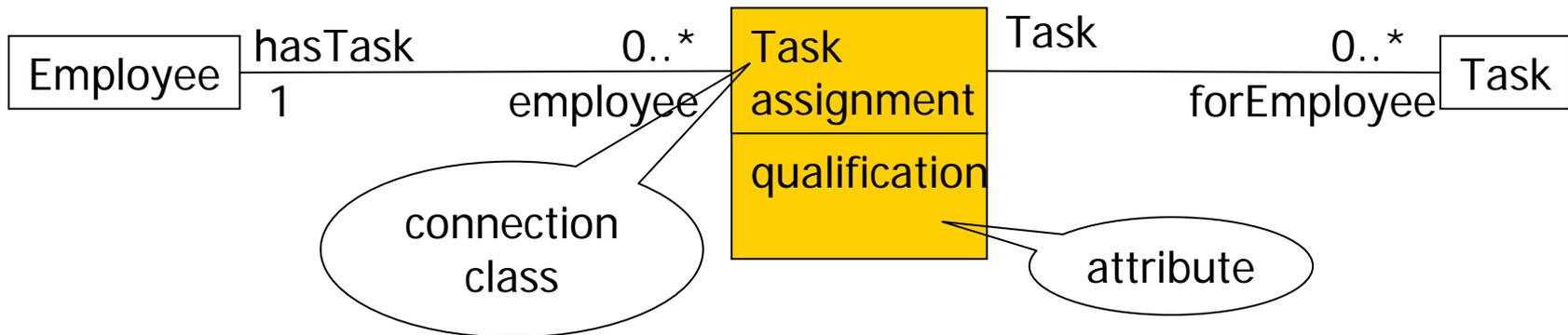




# Different Possibilities for Relationships:

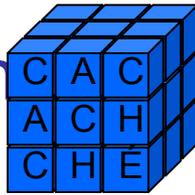
## c) Connection Class

- The relation becomes an own class
  - Suitable for M:N-relations with attributes of their own
  - Two 1:N-relations from each class to connection class
  - In Caché necessary for every many-to-many relation
- Example: Employees - Tasks
  - The employees are assigned to different tasks, e. g. bookkeeping or car service.
  - The qualification attribute states how qualified this employee is for this task.





# Summary: Different Possibilities to Implement Relationships in Caché Jalapeño



- Embedded component class
  - Use `@Embeddable` annotation at component class definition
  - Use for unidirectional relations and private, dependent, encapsulated components
- Persistent component class
  - Just mark the component class as persistent.
  - Use for unidirectional relations and private, dependent, standalone components
- Relationship one-to-many
  - Use `@Relationship` annotation
  - Use for bidirectional relations and private, independent, standalone components
- Relationship parent-to-children
  - Use `@Relationship` annotation
  - Use for bidirectional relations and private, dependent, standalone components
- Connection class
  - Use for bidirectional relations and shared, independent, standalone components
  - Relations with own attributes



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques

## Object Oriented Databases

---

### Object Identity



# Distinguish Objects and Values

---

## Objects

- Example:  
`Room k102 = new Room();`
- are not printable
- must be created and defined
- carry no information itself
- are described by the values of their attributes

## Values

- Example :  
`int i = 182;`
- printable
- already exist
- carry information themselves
- describe objects



## Types of Object Identity

---

- Sum of all values.  
This is not practical because:
  - Values can change
  - Object in DB only contains subset of values of real object
- Values of characteristic attributes
  - E. g. fingerprint or DNA for real persons
  - Difficult to store in DB
  - Not applicable for all classes,  
e. g. chairs in a lecture room



## Types of Object Identity (cont.)

---

- Physical addresses, direct references, pointers
  - Pointers point to component-objects
  - Pointers point to begin of a list for set attributes
- User defined names of a name domain
  - e. g. license numbers of cars
- Automatic keys = surrogate keys = identifier attributes  
(implementation obvious: auto-counter with integer values)
- Abstract objects (implementation hidden to user)



# Object Identity by Physical Addresses

---

- Example in real world: Identify buildings by their geographic position (longitude, latitude)
- On a computer: physical storage address
- Advantages
  - simple implementation
  - efficient access of components
  - address is not a value of the object
- Disadvantages
  - physical data independence violated
  - object cannot be moved
  - Delete an object. Is physical address reusable?



# Object Identity by Names

- Example: Names of persons
- Advantage
  - Name can be structured, easy to read: first- last name.
- Disadvantages
  - can be interpreted as value
  - Name can change (e. g. marriage)
  - It is not possible to have different objects with the same key value, e. g. different persons with the same name.
  - Uniqueness must be checked
  - Problem, when many objects contain the same component object and the key value changes
    - Example: If all vehicles of a branch contain the branch name as a foreign key and the branch name changes all vehicles must be updated.
  - Queries require inefficient joins.



# Object Identity by Surrogate Keys = Identifiers



- Examples: AutoNumber in Access, SEQUENCE in Oracle
- Advantages
  - Uniqueness guaranteed by system
  - cannot be changed
  - carries (nearly) no information, is substitute for object
- Disadvantages
  - Foreign key constraints must be defined
  - Surrugate keys can be interpreted as values.



## Example:

# Vehicles with Identifier-Attributes, O<sub>2</sub> Syntax

```
SET OF (TUPLE OF
 (vehicleID: IDENTIFIER,
 licensePlate: String,
 ...
 model: TUPLE OF
 (model_ID: IDENTIFIER,
 make: String, ...)
 branch: TUPLE OF
 (branch_ID: IDENTIFIER,
 name: String, ...)
 loans: LIST OF (TUPLE OF
 (loan_ID: IDENTIFIER,
 begin: Date, ...)
))
```



## Vehicles with Identifier-Attributes

| vehicle ID | license Plate | ... | model                | branch                            | loans                                               |
|------------|---------------|-----|----------------------|-----------------------------------|-----------------------------------------------------|
| 172963     | UL-E<br>333   |     | 1<br>Mercedes<br>... | 54355<br>Ulm<br>...               | 1000; 1.7.09; ...<br>1020; 5.7.09; ...<br>3000; ... |
| 134584     | B-MA<br>4991  |     | 2<br>BMW<br>...      | 37704<br>Berlin<br>Airport<br>... | 1; 1.8.08; ...<br>20; 6.8.08; ...<br>63; ...        |



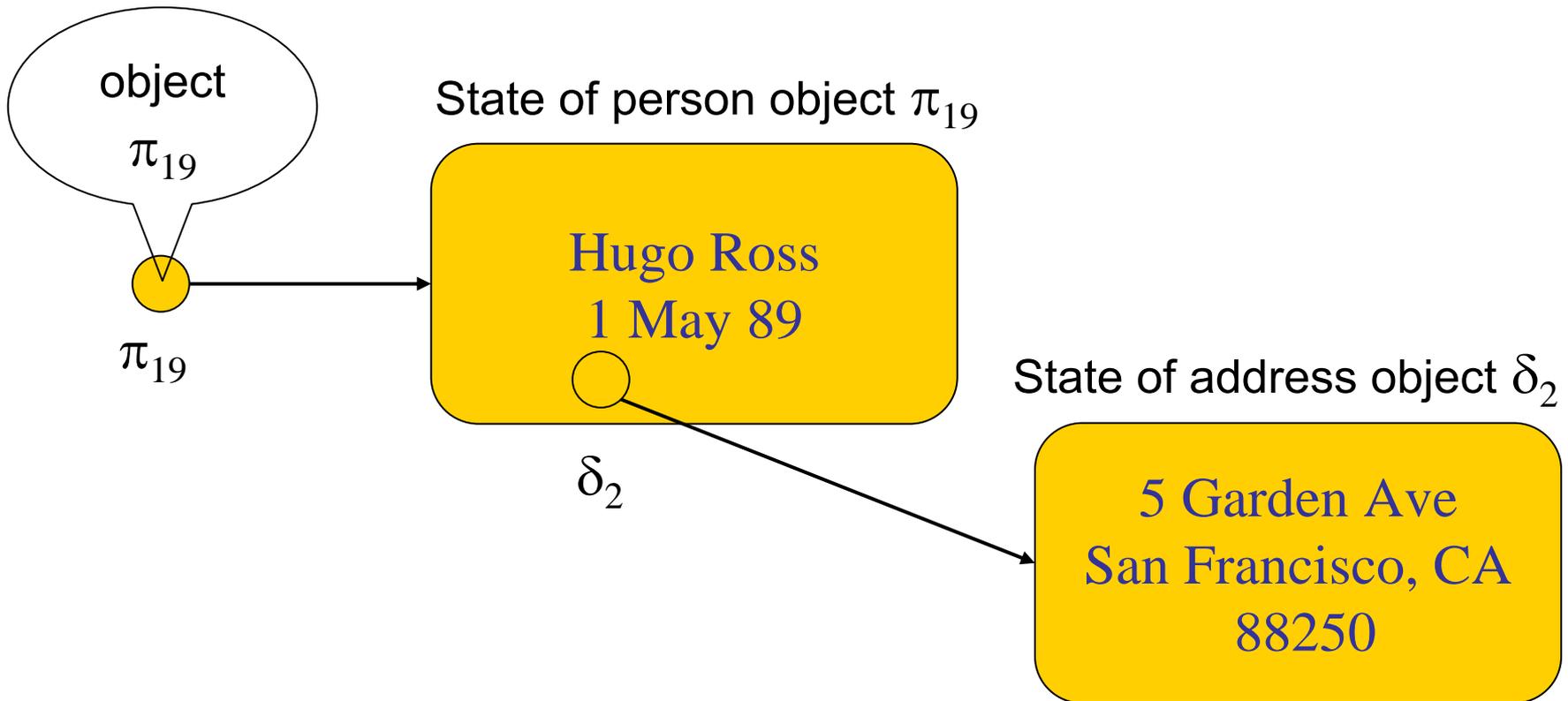
# Object Identity with Abstract Objects

- Objects are elements of a
  - not structured,
  - countably infinite,
  - abstract set
  - with the only information that the elements are different
- Operations on objects
  - Create new object
  - Delete object
  - Test on identity
  - You never can print or see an object itself or its Id
  - You only can use the operations above
- Abstract objects can be implemented (more or less well) by
  - physical addresses (Java)
  - Names
  - Identifier-attributes (Caché)



# Graphical Representation of Abstract Objects Using State-Boxes

A graphical representation is difficult, since objects cannot be printed. Greek characters are just a trial to show abstract objects.





# Abstract Objects

---

- Advantages
  - independent of the implementation
  - theoretically sound
  - foreign key constraints automatically guaranteed
- Disadvantage
  - implemented only in very few real OODBMS
- Two versions
  - One infinite set with abstract objects for all classes
  - Disjoint domains of abstract objects for each class



# Vehicles with Abstract Objects and Disjoint Domains

| Vehicle     | license plate | ... | model        | branch         | loans                                                      |
|-------------|---------------|-----|--------------|----------------|------------------------------------------------------------|
| $\varphi_1$ | UL-E 333      |     | $\mu_1$      | $\beta_1$      | $\lambda_{1000}, \lambda_{1001}, \lambda_{1500}, \dots$    |
| $\varphi_2$ | B-MA 4991     |     | $\mu_2$      | $\beta_2$      | $\lambda_{100}, \lambda_{200}, \lambda_{290}, \dots$       |
| $\varphi_3$ | B-MA 4991     |     | $\mu_2$      | $\beta_2$      | $\lambda_{100}, \lambda_{200}, \lambda_{290}, \dots$       |
| $\varphi_4$ | B-MA 4991     |     | $\mu_{2000}$ | $\beta_{2000}$ | $\lambda_{10000}, \lambda_{20000}, \lambda_{29000}, \dots$ |

$\beta_1$  is not the name of a branch, not the key of a branch, but the whole object of type Branch.

$\varphi_3$  is a shallow copy and  $\varphi_4$  is a deep copy of  $\varphi_2$ . This is explained in more detail on the next slides.



## Operations on Objects

---

- Create new or delete existing object
- Dereference objects to get their values
- Test on identity:  $o_1 == o_2$   
e. g. father of Peter == father of Susan
- Test on shallow equality (of values)
- Test on deep equality (of values)
- Assignment: create a reference to an object and assign it to a variable
- Create shallow copy
- Create deep copy



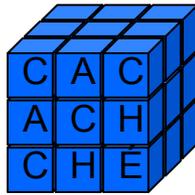
# Comparison of Objects

---

- Philosophical question: When are two objects **identical**?
- Different comparisons
  - Test on **identity**:  
Do two object-references refer to the same object?
  - Test on **equality**:  
Have two possibly different objects the same values?
  - If objects have component objects, compare whether the component objects are identical: **shallow equality**
  - or whether the component objects have the same values: **deep equality**
- Similar alternatives when copying an object:  
shallow copy and deep copy

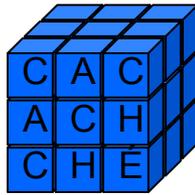


# Object Identity in Caché Jalapeño



- Every persistent class
  - gets constructor-method to create objects in memory
  - and uses insert or save-method of the object manager to store them permanently in DB
- Difference between **Java object reference** and **Caché object ID**
  - An object reference is a temporary identifier for an object in memory.
  - Remove object from memory → object reference is no longer valid
  - cannot be printed
  - An object ID is a permanent identifier for a **stored** object
  - is automatically created by insert- or save-method
  - never changes
  - can be used to locate or delete an object in the DB
  - can be printed
- Caché guarantees correspondence of object references and object IDs:  
View the flash-demonstration of a Java program in single step mode!





# Object Identity in Caché (cont.)

- IDs are class specific
  - ObjectId = Id + ClassName
  - For each class IDs begin with 1.

```
SELECT Id, Name FROM poj os.Branch
```

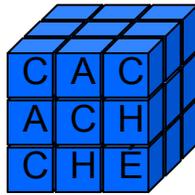
| Id | Name              |
|----|-------------------|
| 1  | Weingarten        |
| 2  | Ravensburg City   |
| 3  | Stuttgart Airport |

```
SELECT Id, Name FROM poj os.Manufacturer
```

| Id | Name     |
|----|----------|
| 1  | BMW      |
| 2  | Mercedes |
| 3  | Suzuki   |



## Object Identity in Caché (cont.)



- Caché Jalapeño does not provide functions for
  - testing shallow equality
  - testing deep equality
  - If needed use the JAVA equals() method.
  - Overwrite the equals() method according to your purposes.  
E. g. two cars are equal if they are of the same model, have the same color, the same date of construction, the same motor ... but of course they cannot have the same license number.
- Creating a copy of an object
  - In Caché Object Script use the method %ConstructClone()
  - Unfortunately this method is not available for Jalapeño.
  - Use the clone() method of JAVA instead and insert the cloned object.  
To use the clone() method you have to implement it.
  - Be careful to create a deep or a shallow copy as required.

## OQL: Object Query Language of the ODMG

### 1.OQL Object Query Language of the ODMG

#### What is OQL?

OQL is a query language for Object Oriented Databases similar to SQL for Relational Databases. Differences and enhancements to SQL are:

- OQL is orthogonal, so for example the result of a query, operator, or function can be used as part of other queries as long as the types fit.
- A dot (".") or an arrow ("->") can be used to access components (implicit join).
- You can access not only properties but also methods.
- OQL not only can query `sets` but also other structures like `bags`, `lists`, `arrays`, and `dictionaries`.
- Instead of insert or update methods you have to use constructor or destructor operations.

#### Overview over OQL

The syntax structure of an OQL select-query is:

**SELECT** defines data to be retrieved and structure of data

**FROM** specifies collections referenced by the query and variables to iterate over those collections

**WHERE** specifies conditions for selecting objects

The following pages present details of OQL.

#### More information:

R.G.G. Cattell. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, San Francisco, California, 1997.

[Open Quasar OQL \(German\)](#)

[ODMG Standard: The Object Query Language \(OQL\)\(PowerPoint\)](#)

## 2.OQL and PATH EXPRESSIONS - Navigation Through Objects



## OQL and PATH EXPRESSIONS

### Navigation Through Objects



- *Path expressions* are used to navigate from one object to another.
- Use dot (".") or arrow ("→")
- Example:  

```
SELECT p.Name.LastName
FROM Persons p
WHERE p→Name→Title = "Dr."
```
- **Equivalent:** FROM p IN Persons
- Persons is extent of persistent class Person
- Result is a multiset of literals; type Bag <string>

112



### 3.OQL: Traversing Attributes and Relationships

- OQL does not allow path expressions to traverse over multivalued attributes and relationships.
- Example: **not allowed**  
SELECT b.Name, b.vehicles.licenseNo  
FROM b IN Branch  
WHERE b.vehicles.color = 'green'
- Use instead variable v for vehicles (continue to next page).

### 3. OQL: Traversing Attributes and Relationships (correct)

- OQL does not allow path expressions to traverse over multivalued attributes and relationships. Use instead variable `v` for vehicles.

- Example: [correct version](#)  
SELECT b.Name, v.licenseNo  
FROM b IN Branch, v IN b.vehicles  
WHERE v.color = 'green'

#### 4.OQL: Extracting from Lists and Arrays

OQL provides an explicit notation for extracting the i-th element from lists and arrays. The first element in any indexed collection is assumed to be 0. OQL also provides the operations first and last

Examples:

`LIST(a,b,c,d)[1] = b`

`LIST (a,b,c,d)[1:3] = LIST (b,c,d)`

`FIRST(LIST(a,b,c,d)) = a`

`LAST(LIST(a,b,c,d)) = d`

A list can be convert into a set: `LISTTTOSET(LIST(1,2,3,2,2)) = {1,2,3}`

The "2" is three times in the list but only once in the set because a set is a collection of unique elements.

If you want to compare OQL list operations with Caché List Operations open lesson "[Type Constructors, Complex Objects](#)" in topic 2.

## 5.OQL: Access Single Elements and Complete Extent

### Access a Single Element

The ELEMENT-operator converts a set with one element into this element: {c} --> c

```
ELEMENT
(
SELECT c
FROM c IN pojos.Car
WHERE c.LicenseNo = 'RV AX-651'
)
```

ELEMENT is implicit in SQL and hence in Caché. If the result of a query is a table with only one column and one row the content of this field is used if appropriate.

### Select all Object of a Class

Example: Select all objects of class Person:

Persons

The same query in SQL:

```
SELECT * FROM Person
```

### Embedded queries

Subqueries are well known in SQL. In OQL you can use subqueries not only in the WHERE-part but also in the SELECT- or in the FROM-part of a query.

Example:

```
SELECT v.model.horsepower
FROM v IN (SELECT v
FROM pojos.Branch b, v IN b.vehicles
WHERE b.address.postcode LIKE '88%')
WHERE v.color = 'green'
```

This is the corresponding query in Caché:

```
SELECT Branch.Vehicle->model->horsepower
FROM pojos.Branch
```

```
WHERE Branch.address_postcode LIKE '88%' AND Branch.Vehicle->color = 'green'
```

## 6.OQL: Grouping

### Structuring Results and Grouping

The function GROUP BY provides explicit reference to the collection of objects within each group or partition.

Example: Group Vehicles by their color and call the set of license numbers property `LicensePlates`. Create a new structure consisting of a brand together with a list of its ATMs.

```
SELECT STRUCT
(
 Color,
 LicensePlates: (SELECT p.licenseNo
FROM p IN PARTITION)
)
FROM c IN pojos.Car
GROUP BY Color: c.color
```

Result type: `struct <string, Set<string>>`

### Grouping and Aggregate Functions

Like SQL OQL provides aggregate operators for collections: min, max, count, sum, avg

Example: List all branches together with the number of its vehicles.

```
SELECT STRUCT
(
 BranchObj: b,
 NrVehicles: COUNT(b.vehicles)
)
FROM b IN Branch
```

Result type: `Struct <Branch, integer>`

## 7.OQL: Quantification

**Existential quantification:** (exists x in e1: cond) is TRUE if at least one element of e1 satisfies cond

Example: Select Branches with at least one green vehicle

```
SELECT b
FROM pojos.Branch b
WHERE EXISTS v IN b.vehicles: v.color = 'green'
```

**Universal quantification:** (for all x in e1: cond) is TRUE if all elements in e1 satisfy cond

Example: Select Branches with only green vehicles

```
SELECT b
FROM Branch b
WHERE FOR ALL v IN b.vehicles: v.color = 'green'
```

### 8.Practice: Define OQL Queries

Now define your own OQL queries:

1. Find all branches which offer at least one car from BMW.
2. Find all managers with the number of employees they supervise; result should be a structure. Managers supervise at least one employee.
3. Find license numbers and brands of all cars belonging to branch "London Westend" which were at least once rent by Mr. Martin Holmes.

Please write down the OQL and the Caché queries for the tasks above. Then compare your solution with the sample solution on the next pages. If you have questions post them in the forum.

## 9.Practice Solution: OQL Queries (1)

**Find all branches which offer at least one car from BMW.**

//OQL

```
SELECT b FROM b IN pojos.Branch
WHERE EXISTS c IN b.vehicles:
c.model.manufacturer.name = "BMW";
```

//Cache

```
SELECT b.%ID FROM pojos.Branch b
WHERE EXISTS (SELECT * FROM pojos.Car c
WHERE b.%ID = c.branch AND
c.model->manufacturer->name = "BMW");
```

## 9.2.Practice Solution: Define OQL Queries (2)

**Find all managers with the number of employees they supervise;  
result should be a structure.**

```
//OQL
```

```
SELECT STRUCT (
Manager: m, NrSubs: COUNT(m.Subordinates))
FROM m IN Employee
WHERE NrSubs > 0
```

```
//Cache
```

```
SELECT m.ID Manager, Count(sub.ID) NrSubordinates
FROM pojos.Employee m, pojos.Employee sub
WHERE sub.Supervisor = m.ID
GROUP BY m.ID
```

or

```
SELECT Supervisor ManagerID, Count(ID) NrSubordinates
FROM pojos.Employee
WHERE Supervisor IS NOT NULL
GROUP BY Supervisor
```

### 9.3.Practice Solution: Define OQL Queries (3)

**Find license numbers and brands of all cars belonging to branch "London Westend" which were at least once rent by Mr. Martin Holmes.**

//OQL

```
SELECT STRUCT (
LicenseNumber: c.licenseNo,
Brand: c.model.manufacturer.name)
FROM c IN pojos.Car, l IN c.loans
WHERE l.customer.firstName = 'John' AND l.customer.lastName = 'Hog' AND
c.branch.name = 'London Westend'
```

//Cache

```
SELECT c.licenseNo, c.model->manufacturer->name Brand
FROM pojos.Loans l
WHERE l.customer->firstName = 'John' AND l.customer->lastName = 'Hog' AND
l.vehicle->branch->name = 'London Westend'
```



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques Object Oriented Databases

---

## Methods



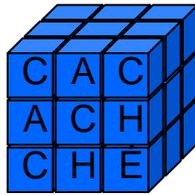
## Methods ...

---

- perform class specific operations
- are a major difference to pure relational model
- allow to combine structural and operational development of an application.
- can be **class methods** or **instance methods**
- can be inherited from superclasses
- Polymorphism: Subclass overrides method inherited from superclass



# Generic Update Methods



| Operation   | Object oriented operation | Operation in Caché                                          |
|-------------|---------------------------|-------------------------------------------------------------|
| Insert      | Constructor               | %New() classmethod<br>%Save() instancem.                    |
| Update      | Access to attribute       | set <i>obj.property</i> = <i>NewVal</i>                     |
| Delete      | Persistent destructor     | %DeleteId( <i>Id</i> )<br>classmethod                       |
| Insert Copy | Copy                      | <u>%ConstructClone(<i>d</i>)</u><br>d: deep or shallow copy |



# Where to Code User Specific Methods? In Database or in Client Application?

---

- Code methods in database classes, if they
  - are used by many different client applications
  - access other persistent objects
  - query database or perform database operations
  - build a unity with the attributes
- Code methods in client application, if they
  - perform time consuming calculations
  - need a large amount of main memory
  - belong to the user interface
  - are invoked concurrently by many clients to use parallel processing on many client computers.



# Practice: Client or Server Side Method



- Where would you code the following methods? Discuss with your neighbor. Find other examples for client side and for server side method.
  - Sum up the amounts of available money for each currency in all ATMs
  - A currency exchange transaction is prepared at home. Calculate the amount of money to be supplied for a certain requested currency value.
  - A CAD application stores data of the developed machines in a database. The engineer can view a drawing of the machine from different perspectives, turn it etc. Calculate the drawing.



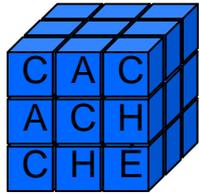
# Methods in Caché



- Code written in ObjectScript:
  - allows use of embedded SQL and
  - macros.
- or in Caché Basic (similar to Microsoft VBScript).
- Set *Language* attribute of a class (for all methods) or individual method(s).
- Can be mixed.
  - ObjectScript can call Basic.
  - Basic can call ObjectScript.
- Both languages compile to same OBJ code.



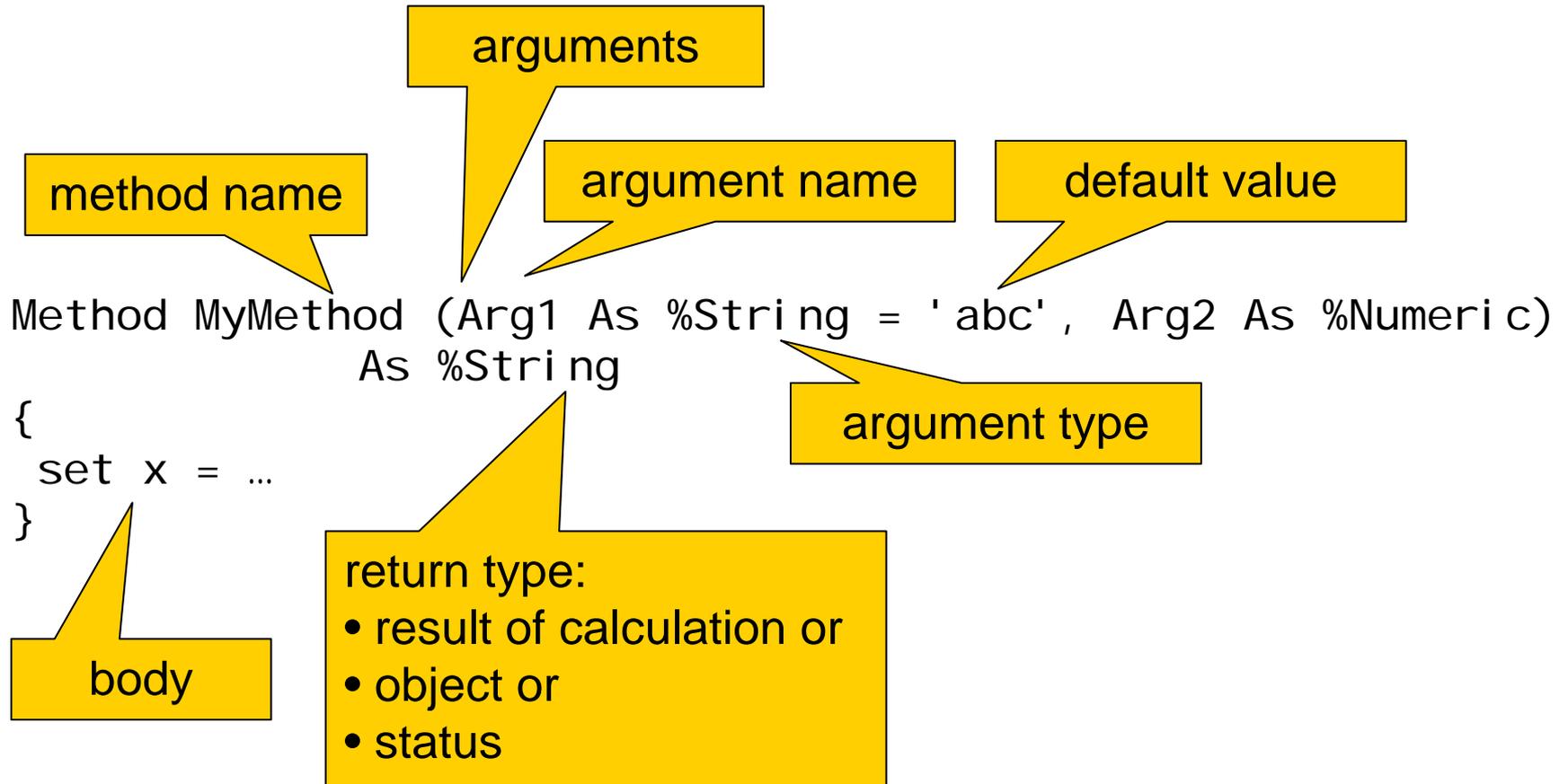
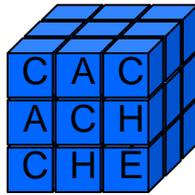
# Inherited Methods



- Classes that extend other classes inherit their methods.
- For example, new persistent classes inherit methods of %Persistent class.



# Method Signature





# Method Arguments



- Arguments are private to method.
- Caller is not required to supply all arguments to a method:  
`do ##class(Package.Class).Method(1,,3,,4)`
- Method must either provide default value for each argument, or use `$data` or `$get` to check argument.  

```
Method Test(a as %String = 1, b as %String
 as %String
{
 if '$data(b) { code here }
}
```



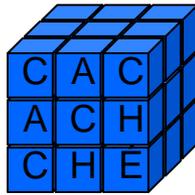
# ObjectScript Arguments



- Pass by value.
  - Default. Use `ByVal` in Code Window to make explicit.
  - Value passed in for input only.
- Pass by reference.
  - Optional. `ByRef` in Code Window required.
    - *Inspector* uses `&` before name.
  - Reference passed in for input and/or output.
  - Optionally, use `Output` in Code Window to document that argument is for output only.
    - Use `*` before name in *Argument* dialog.



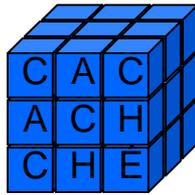
## ObjectScript Arguments (cont.)



- Pass by reference/value is for documentation purposes only.
  - Shows user how to call method.
- Caller, not signature, determines whether argument is passed by reference or by value.
  - Period before argument specifies pass by reference.
- For example:
  - By value: do *oref. Method(a, b)*
  - By reference: do *oref. Method(. a, . b)*



# Objects as Arguments



- Object arguments may be OREFs.
- If method changes properties of object **only**, object argument may be passed by value.
  - Argument inside method references **same** object as calling method.
- If method changes object argument to reference a **different** object, argument must be passed by reference, in order to return new object to caller.



# Method Overloading



- Caché doesn't provide method overloading in a way similar to other languages such as Java.
  - Can't create multiple methods with the same name in the same class.
- To simulate method overloading in Caché:
  - Use \$data/\$get to check arguments.
  - Call different code depending on which arguments are sent.



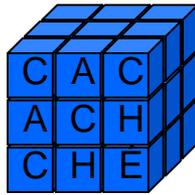
# Private Methods



- Methods are public by default.
  - Methods of other classes can use a public method.
- Set *Private* attribute to *True* to make method private.
  - Only methods in same class or subclasses can use a private method.



# Class and Instance Methods



- *ClassMethod* attribute controls whether method is class or instance method.
  - *True*: Call without an object in memory. Use `##class` syntax.
  - *False*: Call on a **specific** object in memory.
- Examples:

```
set branch = ##class(FCE.Branch).%New()
set status = branch.%Save() // save THIS branch
```
- *ClassMethod=True* **and** *SQLProc=True*: method projected as SQL stored procedure.



# Relative Dot Syntax



- Inside an instance method, use relative dot syntax (..) to refer to another property or method of current instance.
  - Read as “this object’s...”
  - In Caché Basic, use Me.
- Examples:
  - .. *Property*
  - .. *Method()*
    - Instance or class method.
  - .. *#ClassParameter*
    - ObjectScript only.



# Accessor Methods



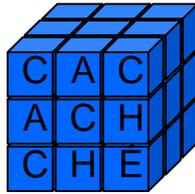
- All properties have hidden “getter” and “setter” *accessor methods*.
  - *PropertyGet()* and *PropertySet()* instance methods.
  - Called automatically when code references property.
  - Not visible in Studio.
- Each pair of statements is equivalent:

*wri te oref. Property*  $\leftrightarrow$  *wri te oref. PropertyGet()*

*set oref. Property = value*  $\leftrightarrow$  *do oref. PropertySet(value)*



# Overriding Accessor Methods



- For special property processing when getting or setting properties, override accessor methods.
  - Last panel of New Property wizard allows this.
- Also used for Computed Properties.
- Override has **no effect** on SQL operations.



# SQL Stored Procedures



- *Stored procedure* is nothing more than a class method that is made available to SQL
- *Stored procedure* may be called from an external tool via ODBC/JDBC.
- Class queries are stored procedures
  - Class queries return result set only.
  - They can't have side effects
  - *SqlProc=True* causes query to be listed in catalog for external tool.
- *Method Stored Procedures* do not return a set of records
  - may have side effects.



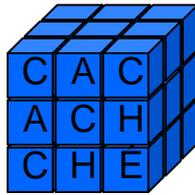
## SQL Stored Procedures (cont.)



- To make class method available as stored procedure:
  - Must be class method.
  - *SqlProc=True*.



## Stored Procedure Code



- Code can be ObjectScript (with or without embedded SQL) or Basic.
- Method can return value and/or have input/output arguments (simple data, streams, or objects).
  - Be careful with return value and argument types.
- Method can also return result set (refer to Query Extent() for an example).



## %sqlcontext Object



- Code should check %sqlcontext to determine whether or not method called as stored procedure.  

```
if $IsObject($get(%sql context)) { code here }
```
- Use properties of %sqlcontext to return information to external tool:
  - SQLCode: SQLCODE error number, or 0 if no error.
  - Message: error message text.
  - RowCount: %ROWCOUNT.



# Calling a Stored Procedure



- Returning result set:

```
call Schema.Table_QueryOrMethodName(arguments)
```

```
call Schema.Table_QueryOrMethodName arguments
```

- Using return value as part of query. For example:

```
select columns
```

```
from tables
```

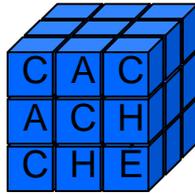
```
where column = Schema.Table_QueryOrMethodName(arguments)
```

- Using ? as placeholder for return value and input/output arguments:

```
? = Schema.Table_QueryOrMethodName(?, ?)
```



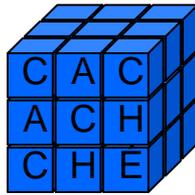
# Instance Methods



- Instance query-methods
  - do not have side effects
  - provide a value of the object
  - can be interpreted as additional property of the object
  - In Caché use computed properties for convenient access by SQL and object code
- Instance update-methods
  - change the status of the object
  - test, whether an operation is allowed
  - In Caché SQL access not possible, use stored procedures instead, with object-id as argument.



# Computed Properties



- Caché supports two types of computed properties/columns.
  - **Always-computed.**
    - Property value computed whenever code references property.
  - **Triggered-computed.**
    - Triggered-computed property/column is stored.
    - Computation triggered when new object is created/inserted, or existing object is changed/updated (properties listed in *SqlComputeOnChange* attribute).
    - **Not** triggered when object is opened, nor when row is selected.
  - Both types compute property/column value based on another property/column (or other properties/columns).



# SqlComputeCode Examples



- Age calculated based on DOB and \$horolog:

```
set {Age}=$select(({{DOB}}' = "") : ((+$h-{{DOB}})\365.25), 1: "")
```

- \$select is a line based if- or case-statement
- This code means

```
if {{DOB}} ' = ""
 {set {Age} = ($piece($horolog, " , " , 1) - {{DOB}})\365.25}
el se
 {set {Age}=""}
```



# SqlComputeCode Attribute



- Specify computation using *SqlComputeCode* attribute.
  - ObjectScript containing *{ColumnName}* reference(s).
    - SQL analog for *..PropertyName* syntax.
    - Unlike regular ObjectScript, **no** extra spaces allowed.
- *{ColumnName}* is:
  - Property's *SqlFieldName* attribute, if present.
  - Otherwise, property's name.



# How To: Create a Computed Property



- Create new property.
- Set *SqlComputed* attribute = *True*.
- Set *Calculated* attribute:
  - For always-computed, set = *True*.
  - For triggered-computed, set = *False*.
- For triggered-computed, optionally specify *SqlComputeOnChange* attribute:
  - List of properties/columns that **trigger** computation.
- Specify *SqlComputeCode*.



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen

# Modern Database Techniques

## Object Oriented Databases

---

Integrity



# Integrity

---

- The object oriented model has inherent integrity constraints.
  - Foreign key constraints are not necessary, due to component and subclass relation
- NOT-NULL-Constraint
  - In Caché use the **Required** attribute for a property
- UNIQUE-Constraint:
  - A combination of attributes must be unique for each object.
  - In Caché use an index with the **Unique** attribute.
  - An Index is also used for access optimization.



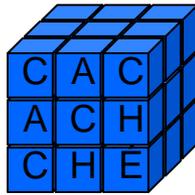
## Integrity (cont.)

---

- CHECK-Constraint
  - In Caché use parameters like **PATTERN**, **VALUELIST**, **MINVAL**, or **MAXVAL**
  - Example Phone: `PATTERN = "3n1"-""3n1"-""4n"`
- DEFAULT-Constraint
  - In Caché use **InitialExpression** attribute of a property
- Use set- and get-methods to access private properties instead of public properties
  - You can code difficult constraint-checks in these methods.

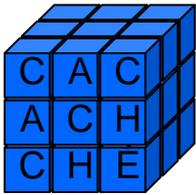


# Callback Methods and Triggers

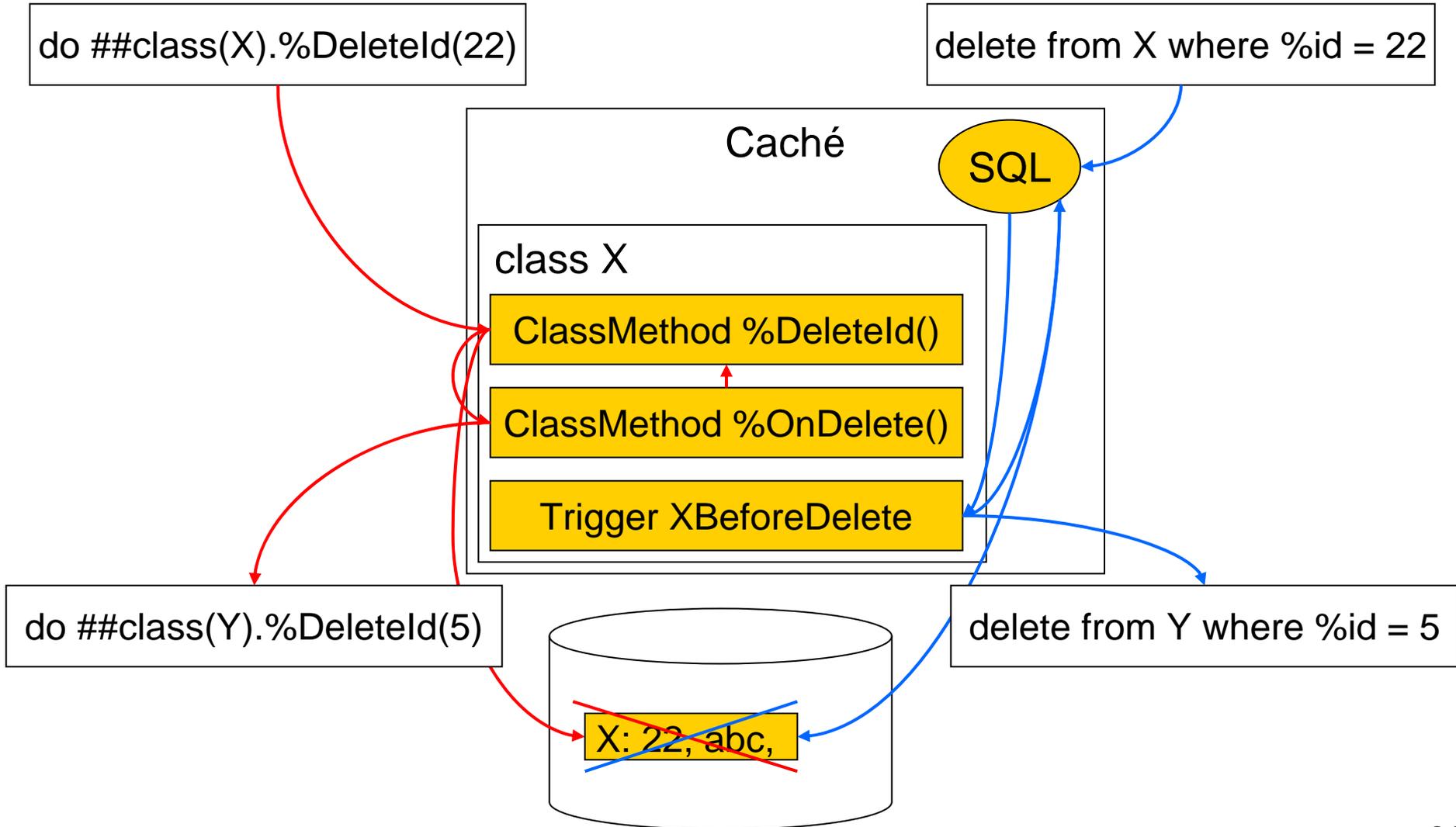


- Use Callback methods or triggers for sophisticated constraint checking or consistency operations.
- Callback methods are used on object side.
- Triggers are used on SQL side.
- Triggers **do not** check object operations!  
Callback methods **do not** check SQL operations!
- To unify the functionality for both OOP and SQL, create class method that callback method and trigger both call.



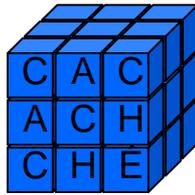


# Callback Methods and Triggers





# OOP Callback Methods



- Events during lifetime of an object can “call back” to methods you supply.
- Each *callback method* can check object and affect outcome of its event.
- Naming convention: `%OnEventName()`.
- Object classes inherit callback method signatures.
  - Override with your code.
- To prevent operation (e.g. prevent saving an object) return an error in the appropriate callback method (e.g. in `%OnValidateObject()`):  

```
quit $$$ERROR($$$General Error, "My error message")
```



# Saving Object



- %Save() calls:
  - Method %OnAddToSaveSet(depth,insert).
    - The *save set* is the set of objects to be saved.
    - Argument: *insert* = 1 for a new object.
    - Return error status to prevent save of object.
    - Use this **pre-validation** callback for creating new objects, modifying properties of existing objects, or adding objects to the save set.
  - Method %OnValidateObject().
    - Use this **pre-validation** callback for custom object validation.
    - Return error status to cause object to fail validation.



## Saving Object (cont.)



- %Save() calls:
  - Method %OnBeforeSave(insert).
    - Use this **post-validation** callback for time-sensitive object changes, such as setting Creation/Edited timestamp properties.
    - Argument: *insert* = 1 for a new object.
    - Return error status to prevent save of object.
  - Method %OnAfterSave(insert).
    - Argument: *insert* = 1 for a new object.
    - Return error status to rollback save of object.
  - Method %OnRollBack().
    - Called when rollback occurs.
    - Return error status to prevent rollback.



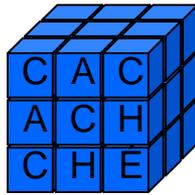
# Deleting Object from Disk



- %DeleteId() calls:
  - ClassMethod %OnDelete().
    - Argument: OID (**not** ID) of object to be deleted. Use \$\$\$oidPrimary macro to get ID from OID.
    - Return error status to prevent deletion.



## Other Callback Methods



| Method                                               | calls Callback Method                                      | Comment                                                                                                         |
|------------------------------------------------------|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| <code>%New()</code>                                  | <code>%OnNew()</code>                                      | Good place for constructor code                                                                                 |
| <code>%OpenId()</code>                               | <code>%OnDetermineClass()</code><br><code>%OnOpen()</code> | Class method<br>Instance method                                                                                 |
| <code>%ConstructClone()</code>                       | <code>%OnConstructClone</code><br>(object, deep)           | <i>object</i> is OREF of cloned object<br><i>deep</i> is <i>deep</i> argument of <code>%ConstructClone()</code> |
| <code>set oref = ""</code><br><code>kill oref</code> | <code>%OnClose()</code>                                    | Removing Object from Memory                                                                                     |



## Practice: Write Callback Method



- A branch must not close before it opens.
- Discuss different possibilities to enforce that constraint.
- Write a callback method that enforces the constraint.



# Practice Solution: Write Callback Method



- A branch must not close before it opens.
- Discuss different possibilities to enforce that constraint.
  - Set MINVAL parameter for Closes greater than MAXVAL parameter for Opens
  - Instead of Closes property Branch gets OpenTime property with  $\text{MINVAL} > 0$ ; implement Closes as computed property  $\text{Closes} = \text{Opens} + \text{OpenTime}$
  - Callback method `%OnValidateObject()`  
**and** Trigger `BranchCheckOpensCloses`



# Practice Solution: Write Callback Method

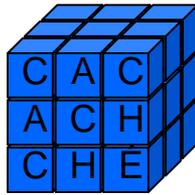


- A branch must not close before it opens.
- Write a callback method that enforces the constraint.

```
Method %OnValidateObject() As %Status[Private]
{
 if (..Closes < ..Opens) {
 quit $$$ERROR($$$General Error,
 "Branch closes before it opens")
 }
 else {Quit $$$OK}
}
```



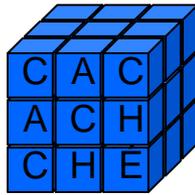
# SQL Triggers



- Events during lifetime of a row can “trigger” methods you supply:
  - Events: Insert, Update, Delete by SQL
- Each *trigger* can check current row and affect outcome of its event.
  - Can also issue additional SQL statements (which may trigger additional methods).
- Trigger code called **after** validation and constraint checking.



## SQL Triggers (cont.)



- Trigger events: Insert, Update, Delete.
- Trigger timings: Before, After.
- For multiple combinations of the same event and timing, specify an *Order* attribute. For example:

```
Trigger T1 [Event = INSERT, Time = AFTER, Order = 1]
{ code here }
```

```
Trigger T2 [Event = INSERT, Time = AFTER, Order = 2]
{ code here }
```

```
Trigger T3 [Event = UPDATE, Time = BEFORE]
{ code here }
```



# Row-Based Triggers



- Caché triggers are row-based rather than statement-based.
- For example, an update of 20 rows will trigger an *after update* trigger for **each** row.
- Some other relational products' triggers would trigger *after update* trigger **once** after updating all rows.



# Writing a Trigger



- Trigger code can use embedded SQL and `{ColumnName}` references to access value of any column in current row.
  - Triggers can't **change** value of columns in current row.
- For UPDATE triggers, code can use:
  - `{column*o}` to access old value.
  - `{column*n}` to access new value.
  - `{column*c}` true if data changed.



# Preventing Event



- To prevent event from occurring:  
set %ok = 0, %msg=" *ErrorMessageText*"
- Used in **before** triggers, event changes never happen.
- Used in **after** triggers, causes rollback of changes.



## Practice: Write Trigger



- A branch must not close before it opens.
- Write a trigger that enforces the constraint.



# PracticeSolution: Write Trigger



- A branch must not close before it opens.
- Write a trigger that enforces the constraint.

create trigger FCE.BranchCheckOpenTime

before insert on FCE.Branch

language OBJECTSCRIPT

```
{if ({Closes*N} < {Opens*N})
```

```
{
```

```
 set %ok = 0
```

```
 set %msg = "Branch closes before it opens"
```

```
}
```

```
}
```



Hochschule  
Ravensburg-Weingarten

Technik | Wirtschaft | Sozialwesen



# Modern Database Techniques

---

## Part 2: Internal Aspects of Caché



# Internal Aspects of Caché



## 1. Advanced storage concepts, performance tuning

1. Storage organization
2. Indices

## 2. System Management

1. Transactions and Locks
2. DBMS read and write processes
3. Log-files = Journals
4. Caché Backup
5. Shadowing

## 3. Security in Databases

1. Authentication: Who may use database
2. Authorization: Which data may a user access
3. Encryption: Additional security
4. Auditing: Track of DB access



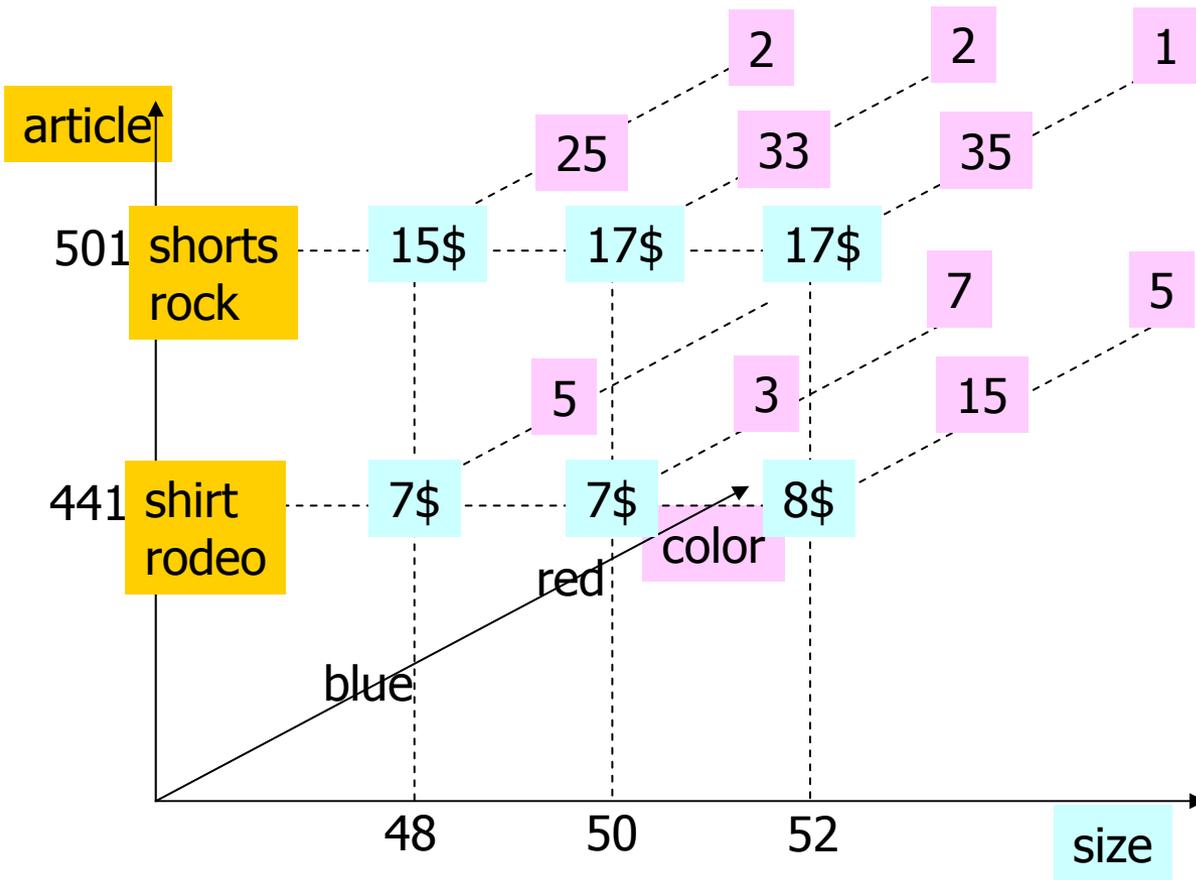
# 1. Storage Organization, Globals

---

- Globals are multidimensional arrays
- Globals only store values at existing positions  
→ no waste of space
- Subscripts may be strings
- All subscripts are concatenated to one string.  
Globals are sorted on that string.



# Storage Organization, Globals



$$\hat{C}_{441} = \text{shirt,rodeo}$$

$$\hat{C}_{441,48} = 7\$$$

$$\hat{C}_{441,48,\text{blue}} = 5$$

$$\hat{C}_{441,50} = 7\$$$

$$\hat{C}_{441,50,\text{blue}} = 3$$

$$\hat{C}_{441,50,\text{red}} = 7$$

$$\hat{C}_{441,52} = 8\$$$

$$\hat{C}_{441,52,\text{blue}} = 15$$

$$\hat{C}_{441,52,\text{red}} = 5$$

$$\hat{C}_{501} = \text{shorts,rock}$$

$$\hat{C}_{501,48} = 15\$$$

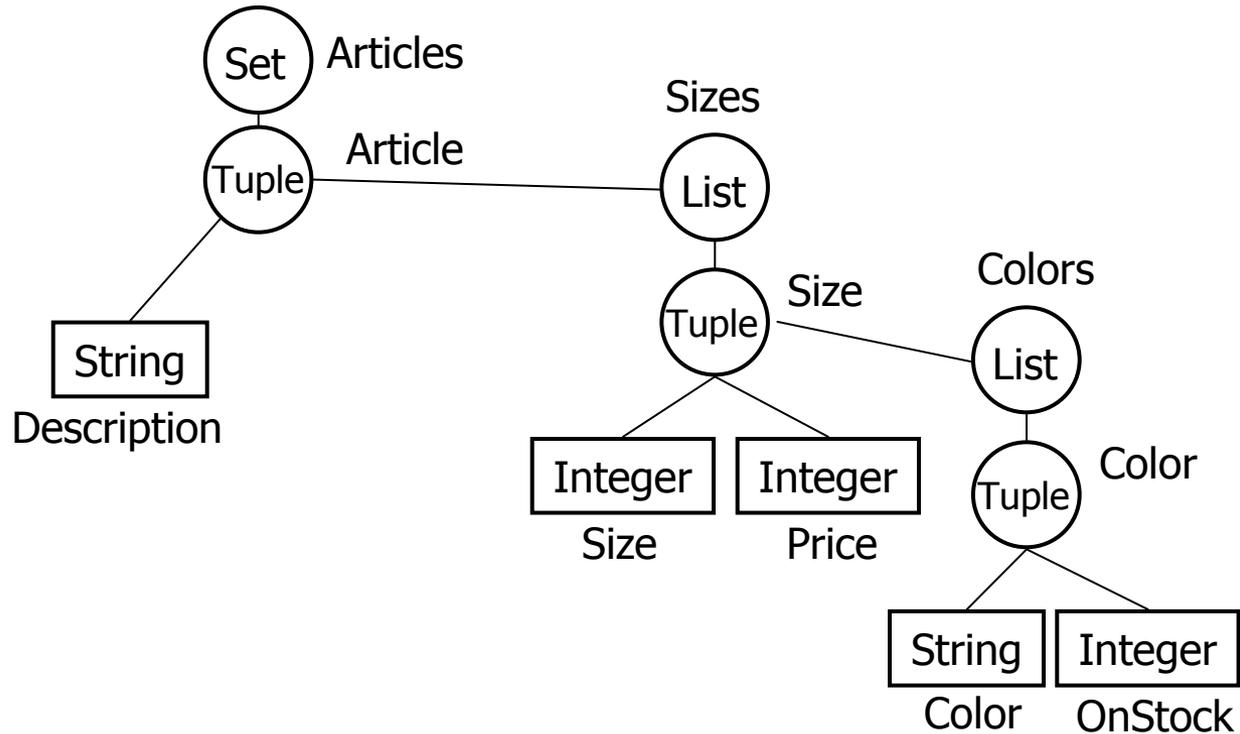
$$\hat{C}_{501,48,\text{blue}} = 25$$

...

Animation: [Caché ObjectScript Tutorial, Ordered Trees](#)



# Object Oriented View of the Example



```
Class C Extends (%Persistent, %Populate)
{Property Description As %String;
 Relationship Sizes As Size
 [Cardinality = children, Inverse = C];
}
```



## Object Oriented View of the Example cont.

---

```
Class Size Extends (%Persistent, %Populate)
{Property Size As %Integer;
 Property Price As %Integer;
 Relationship Colors As Color
 [Cardinality = children, Inverse = Size];
 Relationship C As C
 [Cardinality = parent, Inverse = Sizes];
 Index Key On Size [IdKey, PrimaryKey, Unique];
}

Class Color Extends (%Persistent, %Populate)
{Property OnStock As %Integer;
 Property Color As %String;
 Relationship Size As Size
 [Cardinality = parent, Inverse = Colors];
 Index Key On Color [IdKey, PrimaryKey, Unique];
}
```



# Physical Storage Organization

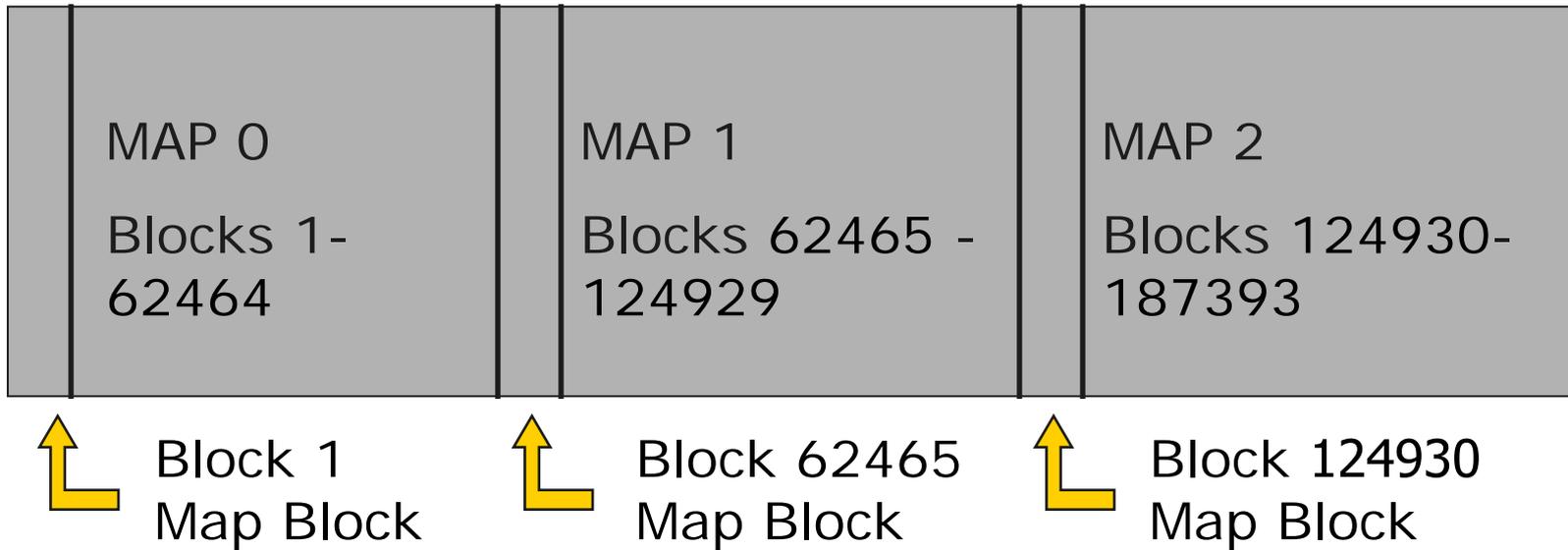
---

- All data stored in file CACHE.DAT
- If one disk is too small for whole database, use up to 7 extension files CACHE.EXT on other disks.
- On UNIX also raw disk partition is possible.
- File is organized in MAPs
- Each MAP has 62464 blocks
- Block size: 8 kByte
- Globals are stored as B\*-tree with subscripts as index.



# Map Blocks

- Each Map consists of:
  - One map block.
    - Map block is always first block of map.
  - 62464 database blocks.





# B\*-tree Storage Structure for Globals



Block 3 Type 9 global dir.  
 Offset 684, link 0  
 ^A 45  
 ^C 44  
 ^FCE.BranchD 40  
 ...

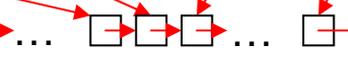
Block 44, Type 66 top pointer, link 0  
 ^C 47 ^C(5446,50,"") 46

Block 47, Type 6 bottom pointer, link 46  
 ^C 286 ^C(9,52,"g") 288 ... 865 ^C(5436,46,"w") 866

Block 46 Type 6 bottom pointer , link 0  
 ^C(5446,50,"") 867 ... 1345 ^C(9996,46,"w") 1346

Block 286, Type 8 data  
 link 288  
 ^C  
 ^C(1) skirt,rodeo  
 ^C(1,46) 10\$  
 ^C(1,46,black) 1  
 ...  
 ^C(9,52,blue) 15

Block 288, Type 8 data  
 link 289  
 ^C(9,52,green) 9  
 ^C(9,52,red) 12  
 ^C(9,52,white) 12  
 ^C(9,52,yellow) 19  
 ...  
 ^C(17,58,red) 19



Block 1346, Type 8 data  
 link 0  
 ^C(9996,46,white) 10  
 ^C(9996,46,yellow) 12  
 ^C(9996,48) 15\$  
 ^C(9996,48,black) 1  
 ...  
 ^C(10000,58,yellow) 15



## Practice:

# Calculate Number of Block retrievals



- How many blocks must be retrieved from memory to find the article with Id 704 for the Clothes example? Use the following information:
  - Assumption: the database has just been started
  - A block pointer for a global entry needs 23 byte
  - A data entry needs 17 byte
  - One object (article) consists of 50 global nodes
  - There are 10 000 articles in the database
  - One block stores 8192 byte of global data
- Immediately after the first search, a second search is started for the article with Id 9999. How many blocks must be retrieved from memory?



# Indices

---

- Additional structures to fasten DB search when the search condition does not depend on the Id.
- Caché stores Indices in globals, hence in B\*-trees.
- Data retrieval with indices:
  - Index does not store a direct reference to a data block.
  - Searching an index results in an Id.
  - This Id is then used to retrieve the object.
- Advantage: No pointers in index must be adjusted, when B\*-tree is reorganized after inserts or deletes



## Indices (cont.)

---

- Different index types
  - Standard index
  - Bitmap index
  - Bitslice index



## A Standard Index ...

---

- associates an ordered set of one or more property values with the object ID values of the object containing the properties.
- stores every object once, even if it has the same value for the index-property as another object.
- is used for properties with many different values, e. g. name of persons.



## Standard Index Example

---

```
Class MyApp.Person Extends %Persistent
{
 Index NameIdx On Name;
 Property Name As %String;
 Property Age As %Integer;
 Property EyeColor As %String;
}
```



## Standard Index Example

---

```
// data
```

```
^MyApp. PersonD = 3
```

```
^MyApp. PersonD(1) = $LB("", "brown", 34, "Jones")
```

```
^MyApp. PersonD(2) = $LB("", "blue", 22, "Smith")
```

```
^MyApp. PersonD(3) = $LB("", "blue", 45, "Jones")
```

```
^MyApp. PersonD(5) = $LB("", "green", 41, "Wolf")
```

```
//index
```

```
^MyApp. PersonI ("NameI dx", " JONES", 1) = ""
```

```
^MyApp. PersonI ("NameI dx", " JONES", 3) = ""
```

```
^MyApp. PersonI ("NameI dx", " SMI TH", 2) = ""
```

```
^MyApp. PersonI ("NameI dx", " WOLF", 5) = ""
```



## A Bitmap Index ...

---

- is used for properties with few different values.
- has one bit-string for each value of the property with one bit for every object.
  - 1 is stored, if the object has the specific value
  - 0 is stored otherwise
- If there are more than about 64000 objects, the bit-strings are divided in so called **chunks**
- Abbreviations are used for all 1 or all 0
- If a bitmap index is defined for a class automatically an **extent bitmap index** is maintained: It stores 1 if the Id exists, and 0 if not.



## Bitmap Index (cont.)

- Special functions support the usage of bitmap indices

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| <b>\$Bit</b>      | Set or get a bit within a bit string.                            |
| <b>\$BitCount</b> | Count the number of bits within a bit string.                    |
| <b>\$BitFind</b>  | Find the next occurrence of a bit within a bit string.           |
| <b>\$BitLogic</b> | Perform logical (AND, OR) operations on two or more bit strings. |

- They are used to accelerate SQL-queries like  
`SELECT COUNT(*) FROM Person WHERE EyeCol or = "bl ue";`



# Bitmap Index Example

```
In MyApp.Person add:
Index EyeI dx
On EyeCol or
[Type = bi tmap];
```

| Id | Name  | Age | EyeColor |
|----|-------|-----|----------|
| 1  | Jones | 34  | brown    |
| 2  | Smith | 22  | blue     |
| 3  | Jones | 45  | blue     |
| 5  | Wolf  | 41  | green    |

```
// i ndex gl obal
```

```
^MyApp. PersonI ("EyeI dx", "bl ue", 1) = 01100...
```

```
^MyApp. PersonI ("EyeI dx", "brown", 1) = 10000...
```

```
^MyApp. PersonI ("EyeI dx", "green", 1) = 00001...
```

```
// extent i ndex gl obal
```

```
^MyApp. PersonI (" $Person", 1) = 11101...
```

Chunk



# Demonstration: Indices and Performance



- In the Clothes example query C for a certain Id  
`SELECT * FROM C where Id = 777`
  - Query is fast, 0.001 s
- Query C for a certain description  
`SELECT * FROM C where Description = 'ski rt, rodeo'`
  - Query is slow, 0.320 s
  - View query plan
- Define a Standard Index on Description
  - Rerun query
  - Query is faster, 0.150 s
  - Compare query plan



# Demonstration: Indices and Performance cont.



- Drop the Standard Index
- Define a Bitmap Index on Description
  - Rerun query
  - Query is executed faster again: 0.090 s
  - Compare query plan



## Practice: Discuss Indices

---



- In which cases should you use standard indices,
- in which cases should you use bitmap indices?
- Give Reasons for your statements.



# Practice Solution: Choosing Bitmap vs. Standard



- Think of it as an area question.
- Bitmaps are short and fat, the more unique values the taller they become, the more rows the fatter they become:
  - Both indices have the same number of rows, but the one with 4 distinct values for the index columns is taller.

2 values

4 values

Nr. of distinct values for index columns



Nr. of rows in table  
= Nr. of bits in index



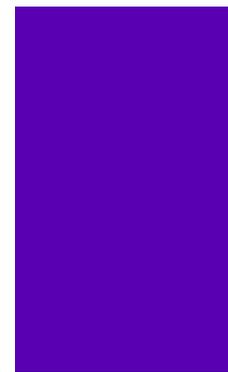
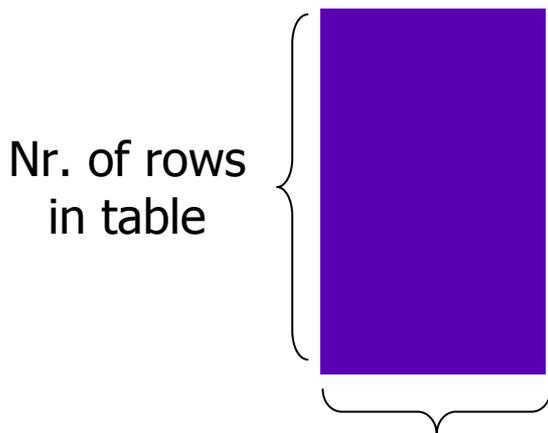
# Practice Solution: Choosing Bitmap vs. Standard contd.



- Standard indexes are tall and skinny, the more rows in the table the taller the index:
  - As the number of distinct values change the size of a Standard index does not change much.

2 values

4 values



(almost) fixed number of bits  $\approx 20$



# Practice Solution: Choosing Bitmap vs. Standard contd.



- At some point the bitmap will become so tall the overall area will be greater than the standard index and the standard index will be faster.

Bitmap



Standard





# Practice Solution: Choosing Bitmap vs. Standard contd.



- Estimate number of records in table  
= number of objects in class
- Determine the number of distinct values for the index columns.
- Calculate the sizes of standard and bitmap index.
- Choose the smaller index.
- Bitmap index not allowed for unique index.
- There are some things Bitmaps are very good for.
  - SELECT Count....
  - complex WHERE clause with AND and OR



## Bitslice Index

---

- A way to index numbers so that they can be summed or averaged quickly.
- No compound Bitslice indices.
- The value is broken down into its binary value and then indexed on each bit of that value.
- 10-20x slower updating than standard or bitmap indices.



## How does a Bitslice Index work?

---

- Take a numeric value (e.g. Weight, Total Bill).
- Scale it to an integer, using the property scale parameter.
- Use the binary value of the integer.
- Store each bit as condition in a bitmap.



## Sample Bitslice Index 1

---

- If the weight for one record is 0.83 tons, this is scaled to 83, then broken down into the binary value '1010011' and the record is indexed under:
  - Weight bit 7, 5, 2 and 1.



## Sample Bitslice Index 2

- Customer #3 has a bill of \$27 (binary 11011) and Customer #4 has a bill of \$17 (binary 10001).

Record number

|               | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | ... | 10million | ... |
|---------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|-----|-----------|-----|
| 2) Sex = Male | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0  | 1  | 0  | 1  | 1  | 0  | 1  | ... |           |     |
| 3) Bill bit 5 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1  | 0  | 1  | 0  | 0  | 1  | 0  | ... |           |     |
| 4) Bill bit 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0  | 0  | 1  | 0  | 0  | 0  | 1  | ... |           |     |
| 5) Bill bit 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0  | 0  | 1  | 0  | 1  | 1  | 1  | ... |           |     |
| 6) Bill bit 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0  | 0  | 1  | 0  | 1  | 1  | 1  | ... |           |     |
| 7) Bill bit 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 1  | ... |           |     |

A red oval highlights the columns for record numbers 3 and 4 across all rows.



# Sample Bitslice Index 2 contd.

- To calculate total bill for all Males:
  - $(\text{cond2 AND cond3}) * 16 + (\text{cond2 AND cond4}) * 8 + (\text{cond2 AND cond5}) * 4 + \dots$

Record number

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 .... 10million ....

|               |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |
|---------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|
| 2) Sex = Male | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | ... |
| 3) Bill bit 5 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | ... |
| 4) Bill bit 4 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | ... |
| 5) Bill bit 3 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | ... |
| 6) Bill bit 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | ... |
| 7) Bill bit 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ... |



# Sample Bitslice Index 2 contd.

- To calculate total bill for all Males:
  - $(\text{cond2 AND cond3}) * 16 + (\text{cond2 AND cond4}) * 8 + (\text{cond2 AND cond5}) * 4 + \dots$

Record number

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 .... 10million ....

|               |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |
|---------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|
| 2) Sex = Male | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | ... |
| 3) Bill bit 5 | 0 | 1 | 1 | 0 |   | 0 | 0 |   |   | 0 |   | 0 | 0 | 0 | 0 |   | ... |
| 4) Bill bit 4 | 0 | 1 | 0 | 1 |   | 1 | 0 |   |   | 0 |   | 0 | 0 | 0 | 1 |   | ... |
| 5) Bill bit 3 | 0 | 0 | 0 | 1 |   | 0 | 1 |   |   | 0 |   | 0 | 1 |   | 1 |   | ... |
| 6) Bill bit 2 | 0 | 1 | 0 | 1 |   | 0 | 1 |   |   | 0 |   | 0 | 1 |   | 1 |   | ... |
| 7) Bill bit 1 | 0 | 1 | 1 | 1 |   | 0 | 0 |   |   | 0 |   | 0 | 0 | 0 | 1 |   | ... |



## 2. System Management

---

- Review standard system tasks of DBMS
- Emphasis on specialties of Caché
- Topics:
  1. Transactions and Locking
  2. DBMS Read and Write-Process
  3. Log-Files = Journals in Caché
  4. Caché Backup
  5. Shadowing



## 2.1 Transactions and Locks

---

- Given a series of related operations that alter database, *transaction processing* (TP) guarantees that either all operations happen, or none do.
- Precede operations with the command to *Start a transaction*.
- Follow operations with the command *to Commit the transaction*.
- If hardware fails during transaction, operations so far are automatically *rolled back*.
- Locks guarantee exclusive access to data



# Transaction Processing

---

- OOP operations and SQL operations are automatically transactions.
  - Saving or deleting individual objects.
  - Inserting, updating, or deleting rows.
  - Use transaction processing commands to guarantee a series of these operations.
- The next slide shows ObjectScript and SQL transaction processing commands.
  - Generally incompatible.
  - Use ObjectScript TP commands for OOP.
  - Use SQL TP commands for SQL.



# Transaction Processing Commands

| Action                              | ObjectScript | SQL                                                        |
|-------------------------------------|--------------|------------------------------------------------------------|
| Start transaction                   | tstart       | START TRANSACTION                                          |
| Commit transaction                  | tcommit      | COMMIT                                                     |
| Rollback transaction                | trollback    | ROLLBACK                                                   |
| Create savepoint inside transaction | tstart       | SAVEPOINT <i>Name</i>                                      |
| Rollback nested transaction         | trollback 1  | ROLLBACK TO SAVEPOINT<br>ROLLBACK TO SAVEPOINT <i>Name</i> |



# Concurrency Control

- To prevent multiple users from editing same persistent object at same time, use *concurrency* options.
  - Uses Caché Lock Table.
- Method 1: Use Lock command:  
`Lock ^FCE.BranchD(7)`
- Method 2: %OpenId() method's second argument:  
`set branch = ##class(FCE.Branch).%OpenId(7, 4)`

| Lock level  | 0       | 1                                | 2                                | 3                                     | 4              |
|-------------|---------|----------------------------------|----------------------------------|---------------------------------------|----------------|
| Description | No lock | shared lock during load and save | shared lock during load and save | shared lock while object is in memory | exclusive lock |



# Concurrency Control (cont.)



- Decrease an object's concurrency:  
do branch. %DowngradeConcurrency(3)
- Increase an object's concurrency:  
do branch. %UpgradeConcurrency(4)
- Use %LockId() and %UnlockId() methods to specify concurrency without opening object.  
do ##class(FCE.Branch). %LockId(7, 3)  
do ##class(FCE.Branch). %UnlockId(7, 3)
- %LockExtent() and %UnlockExtent() methods allow specification of concurrency for entire extent.  
do ##class(FCE.Branch). %LockExtent(3)  
do ##class(FCE.Branch). %UnlockExtent(3)



# Versioning

---

- A class can automatically maintain property containing integer version number for each of its objects.
- When object is filed, version number is incremented.
- Used to implement **optimistic concurrency**:
  - Try to do your work and hope, that no other user disturbs it.
  - When last object is saved a concurrency error may occur and all work is rolled back.



# Using Versioning

---

- If two processes open same object for editing, **without** specifying concurrency argument:
  - First process saves object, which increments version number.
  - Second process attempts to save object, but version number doesn't match version number of object on disk, so save fails.
  - Application can take action to resolve failure, including updating version number.
- SQL Update also checks version number.



## How To: Specify Versioning Property

---

- Create an %Integer property (usually named *Version*, but any name is permitted).
- Set the value of the property's InitialExpression keyword to 0.
- Set the *VERSIONPROPERTY* class parameter equal to the property name.



## 2.2 DBMS Read and Write-Process Write Image Journal File

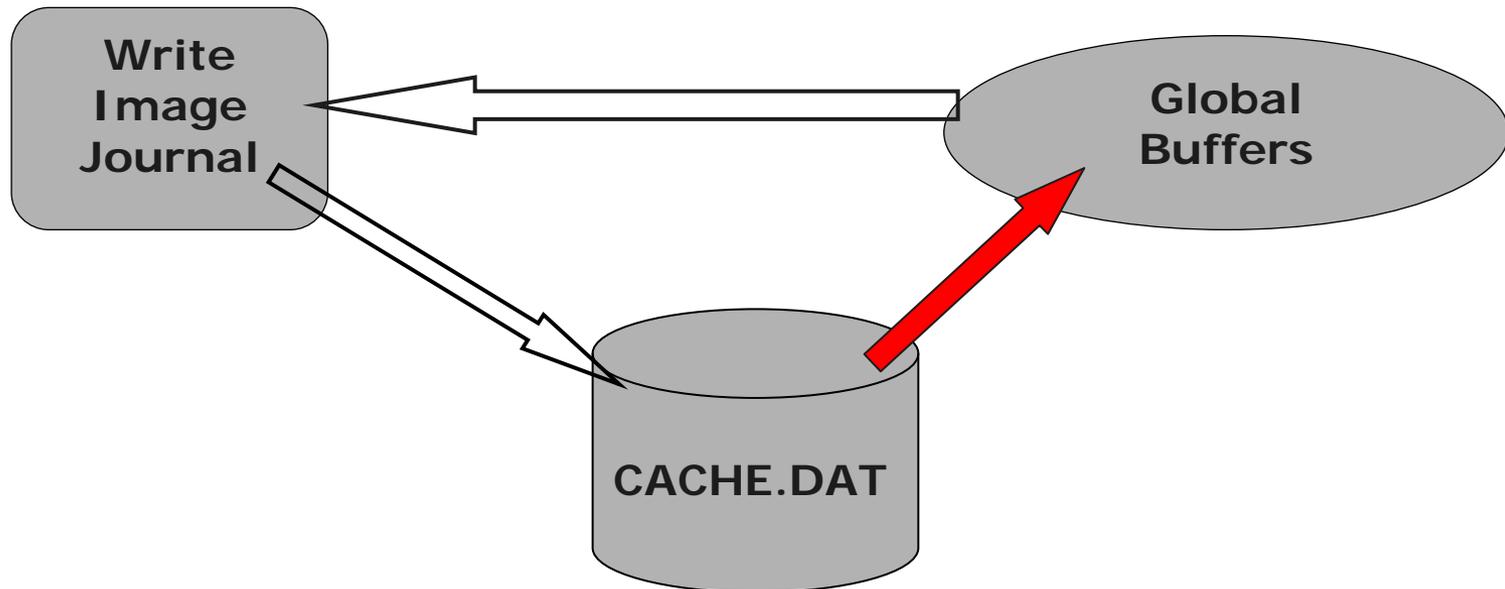
---

- CACHE.WIJ
- Holds operations of all not completed transactions.
- File is small initially and expands as necessary.
  - Need enough file system space for WIJ to expand to size of global cache.
- If WIJ file cannot be created during startup, system will not start.
- Operated by WRTDMN = write demon process



# WIJ During Database Writes

- Process requests routine or global.
  - First,
    - Caché retrieves and places into buffer.
  - Next,
    - Process accesses information in buffer.

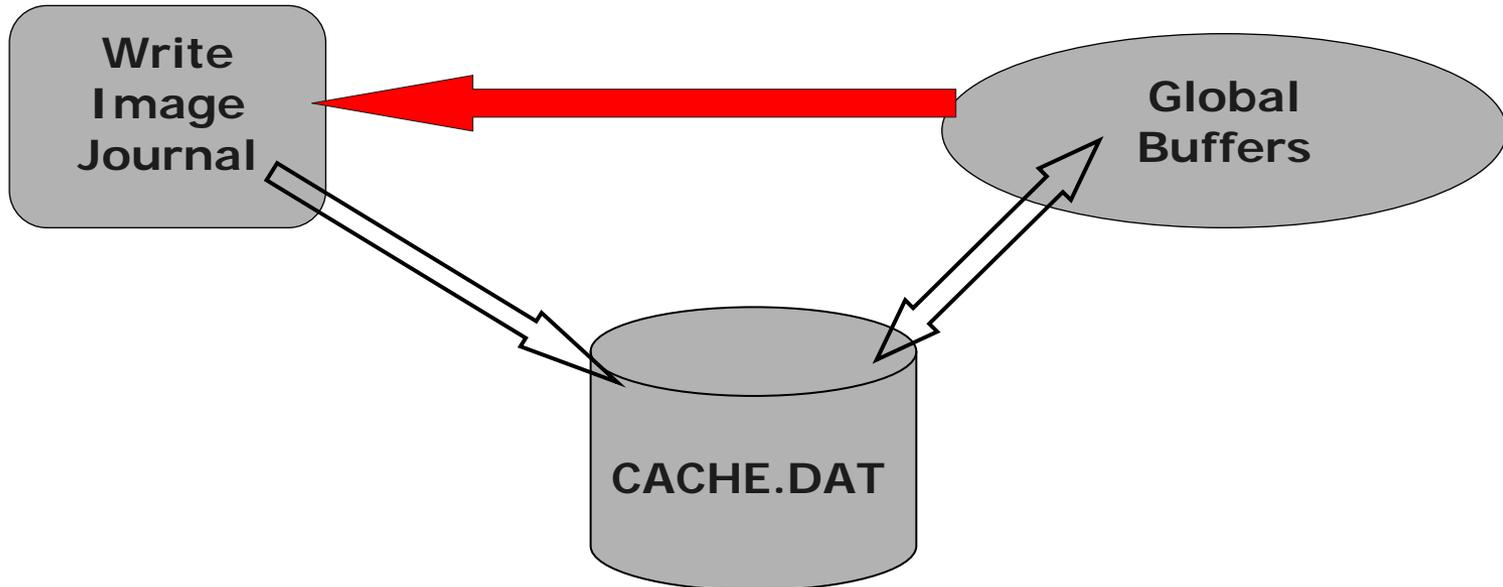




# WIJ During DB Writes

## Two-Phase Write Protocol, 1st Phase

- Process changes routine/global.
  - First,
    - Write daemon wakes.
    - Write daemons writes changed buffers into WIJ.
    - Flag is set to mark WIJ active.

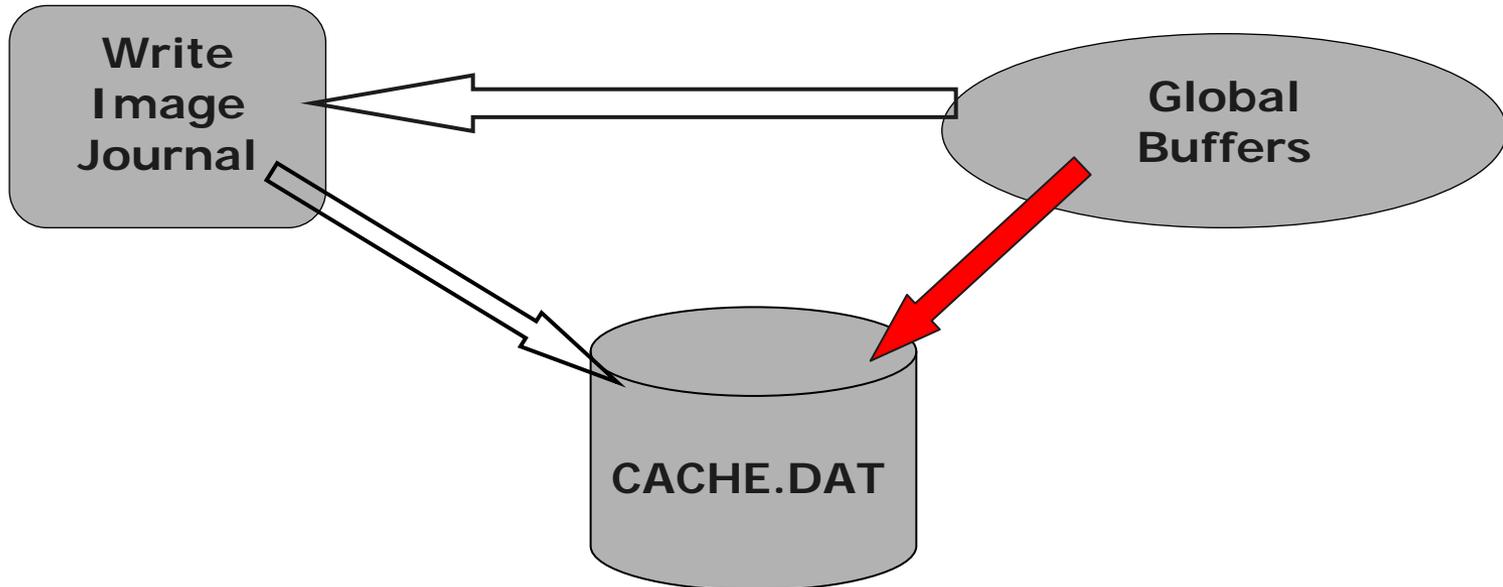




# WIJ During DB Writes

## Two-Phase Write Protocol, 2nd Phase

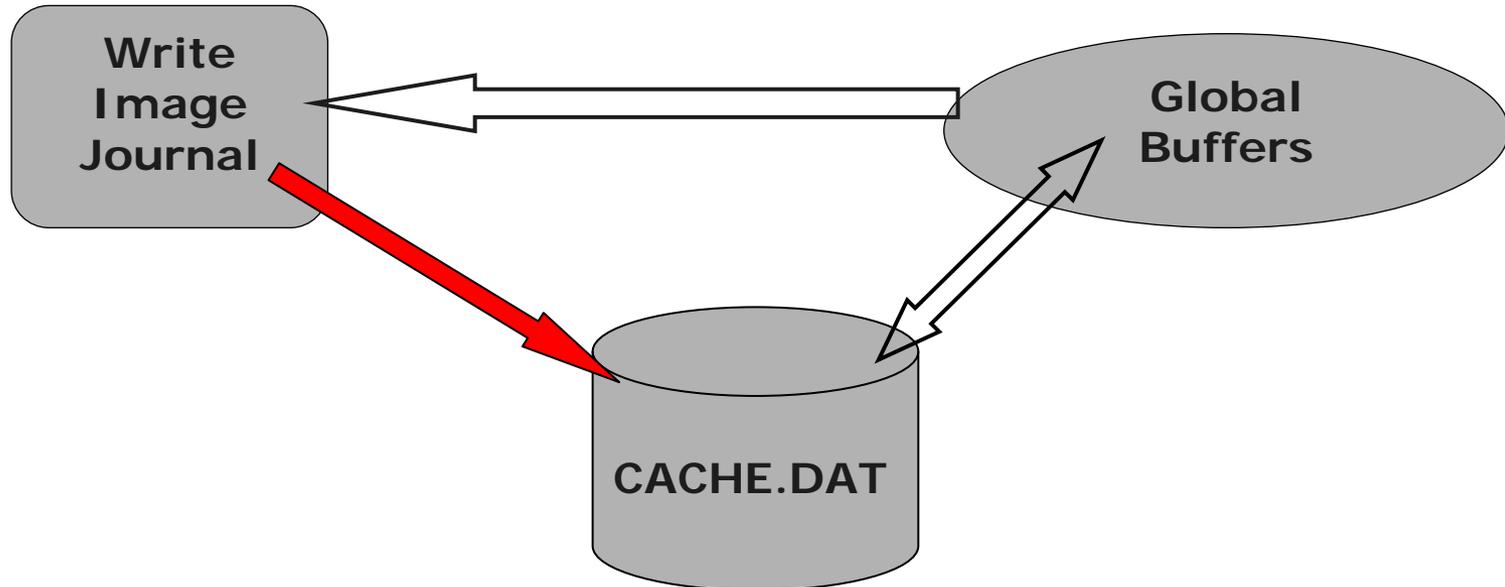
- Process changes routine/global (cont.).
  - Next,
    - Caché writes information into database from buffers.
    - Flag is reset to mark WIJ deleted.
    - Write daemon sleeps.





# WIJ During Recovery

- If abnormal shutdown detected upon start, Caché automatically rewrites any updates in WIJ to database.
  - WIJ is rolled forward.





## 2.3 Logging = Journaling in Caché

---

- Log-File = Journal = Record of changes made to database between backups.
- Operated by JRNDMN = journal demon process
- Supports restoring database from backup.
  - Referred to as roll forward recovery.
- Supports transaction processing.
- Provides crash resiliency with WIJ.
  - Allows recovery of journal records and rollback.
- Required for:
  - Shadowing
  - Cluster configurations



# Global Journal State

[ Configuration ] > [ Local Databases ] > [ Database Properties ]

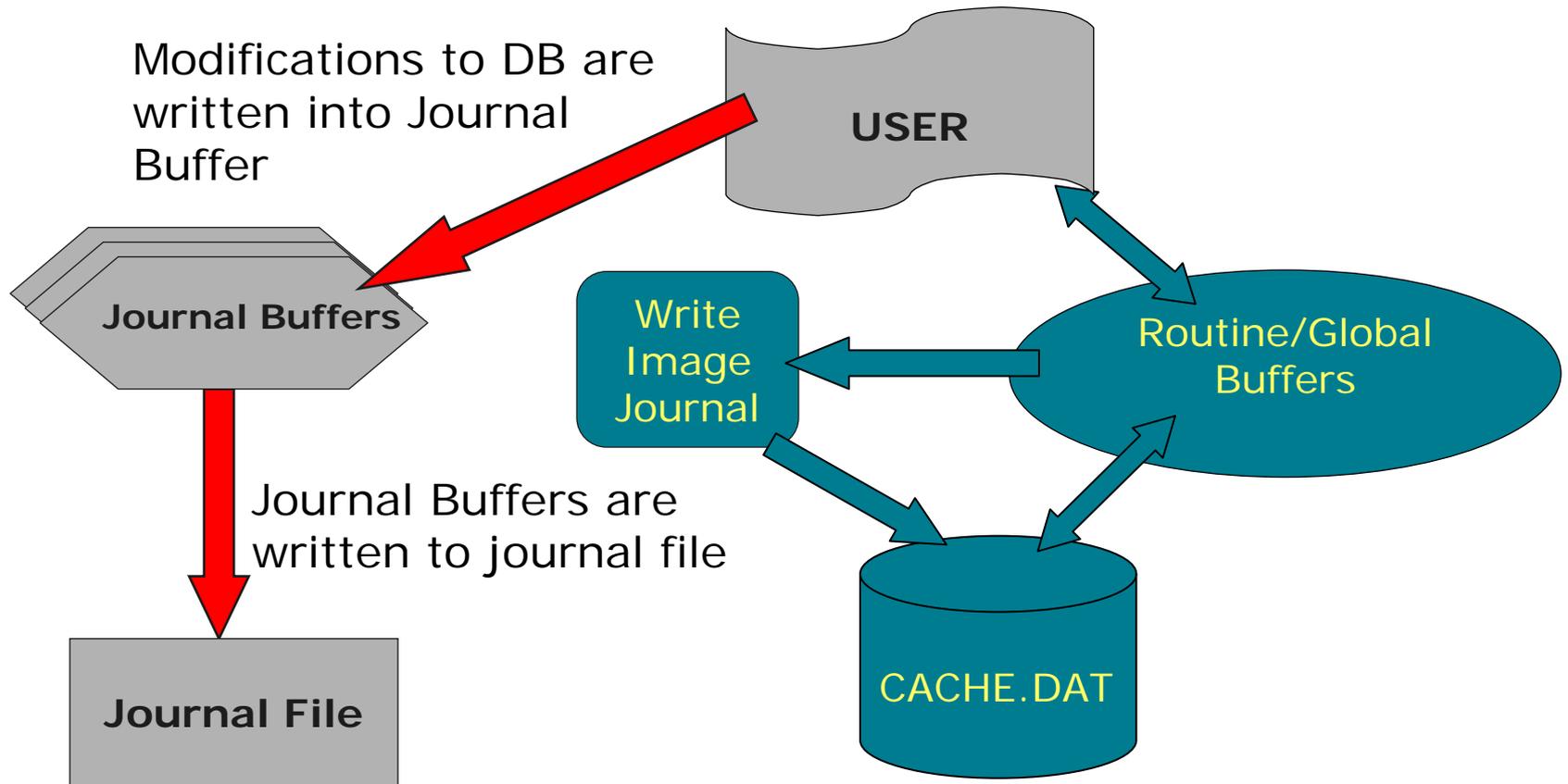
- Yes.
  - Journals all globals in database.
  - Default.
  - Recommended.
- No.
  - No globals journaled.

Edit Database Properties: TEST

|                                       |                       |
|---------------------------------------|-----------------------|
| Name:                                 | TEST                  |
| Directory:                            | C:\Cache825Min\Mgr\TE |
| Block Size (Bytes):                   | 8192                  |
| Size (MB):                            | 1                     |
| Maximum Size (MB):                    | 0 (0 for Unlimited)   |
| Expansion Size (MB):                  | 0 (0 for Default)     |
| Resource Name:                        | %DB_%DEFAULT          |
| Global Journal State:                 | Yes                   |
| New Global Collation:                 | Cache standard        |
| New Global Growth Block:              | 50                    |
| New Global Pointer Block:             | 16                    |
| Preserve global attributes on delete: | No                    |
| Read Only?                            | No                    |
| Mount Required at Startup?            | No                    |



# Journaling Flow





# Configuring Journaling

[ Home ] > [ Configuration ] > [ Journal Settings ]

|                                |                                                                                           |           |
|--------------------------------|-------------------------------------------------------------------------------------------|-----------|
| Journal directory:             | D:\RNdirectory                                                                            | Browse... |
| Alternate journal directory:   | E:\RNdirectory                                                                            | Browse... |
| Start new journal file every:  | 1024 (MB)                                                                                 |           |
| When to purge journal files:   | <input type="radio"/> After this many days 2 (1 - 100)                                    |           |
|                                | <input checked="" type="radio"/> After this many successive successful backups 2 (1 - 10) |           |
| Freeze on error:               | No                                                                                        |           |
| Write image journal directory: | D:\WIJdirectory                                                                           | Browse... |

- *Journal directory*
  - Default is <installdir>\mgr\journal.
  - Move to isolated disk, if possible.
- *Alternate journal directory*
  - Alternate directory used when primary fills.
  - Continues to use alternate unless manually changed.



# Journal File

- Name is `yyyymmdd.nnn`.
- Contains:
  - Sets and kills for globals.
  - Transaction begins and commits.
  - Journal markers for switch.

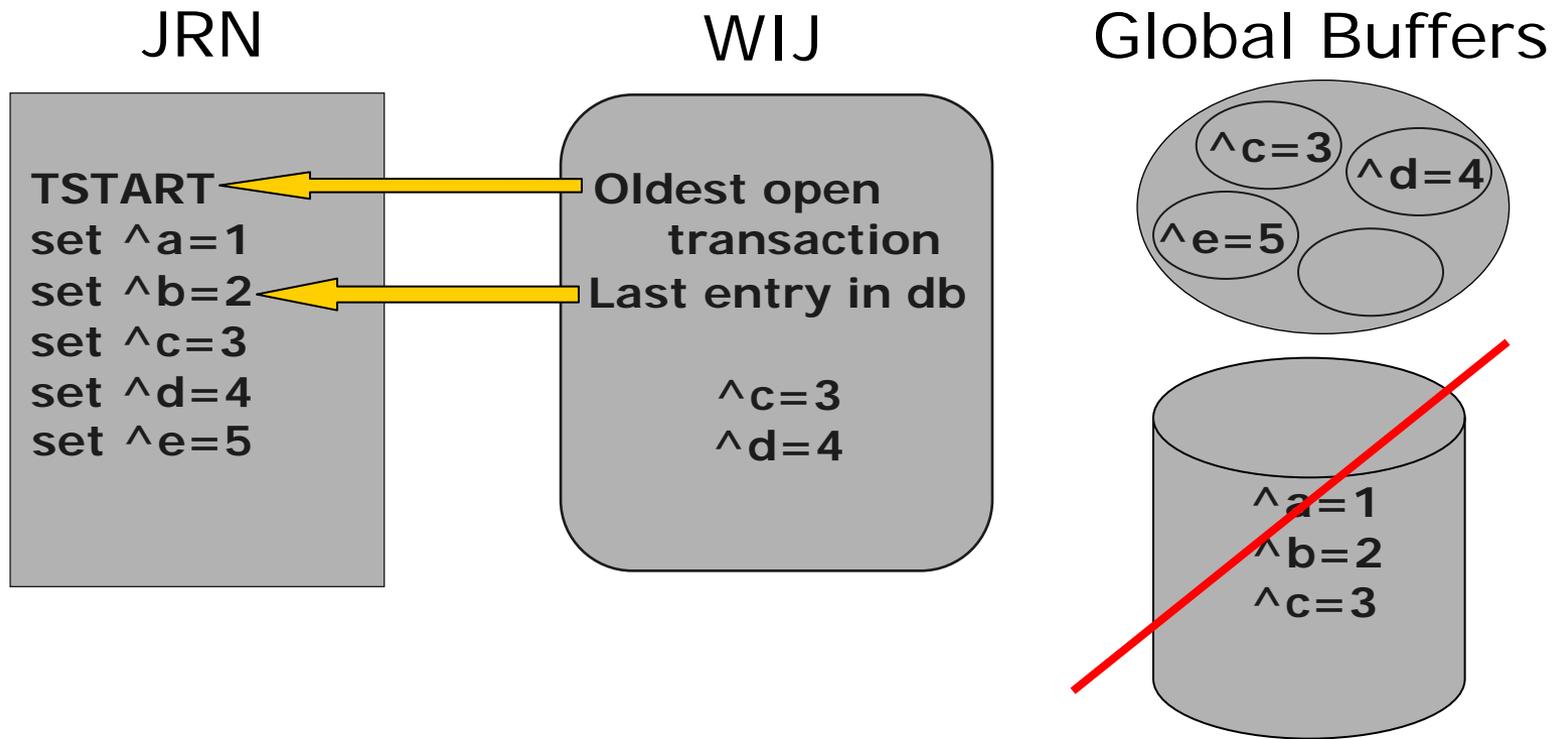
[Home] > [Journals]

| Offset                 | Time                       | Proc       | Type               | InTxn      | Global                                       | Database   |
|------------------------|----------------------------|------------|--------------------|------------|----------------------------------------------|------------|
| <a href="#">196644</a> | 2006-01-19 19:47:21        | 908        | SET                | No         | CacheAuditD("OLEARYD410: CACHE826MIN2",1701) | c:\cache82 |
| <a href="#">196724</a> | 2006-01-19 19:47:21        | 4216       | SET                | No         | %cspSession(0,5208627809,"16ltZroe00")       | c:\cache82 |
| <a href="#">196792</a> | 2006-01-19 19:47:21        | 4216       | KILL               | No         | %cspSession(0,5208627741,"16ltZroe00")       | c:\cache82 |
| <a href="#">197120</a> | 2006-01-19 19:47:21        | 4216       | SET                | No         | %cspSession("16ltZroe00")                    | c:\cache82 |
| <a href="#">197184</a> | 2006-01-19 19:47:21        | 4216       | SET                | No         | %cspSession(0,5208627824,"16ltZroe00")       | c:\cache82 |
| <a href="#">197252</a> | 2006-01-19 19:47:21        | 4216       | KILL               | No         | %cspSession(0,5208627809,"16ltZroe00")       | c:\cache82 |
| <a href="#">197580</a> | 2006-01-19 19:47:21        | 4216       | SET                | No         | %cspSession("16ltZroe00")                    | c:\cache82 |
| <a href="#">197904</a> | 2006-01-19 19:47:21        | 4216       | SET                | No         | %cspSession("16ltZroe00")                    | c:\cache82 |
| <b>197968</b>          | <b>2006-01-19 19:47:21</b> | <b>928</b> | <b>BeginTrans</b>  | <b>Yes</b> |                                              |            |
| <b>197984</b>          | <b>2006-01-19 19:47:21</b> | <b>928</b> | <b>CommitTrans</b> | <b>Yes</b> |                                              |            |



# Crash Resiliency

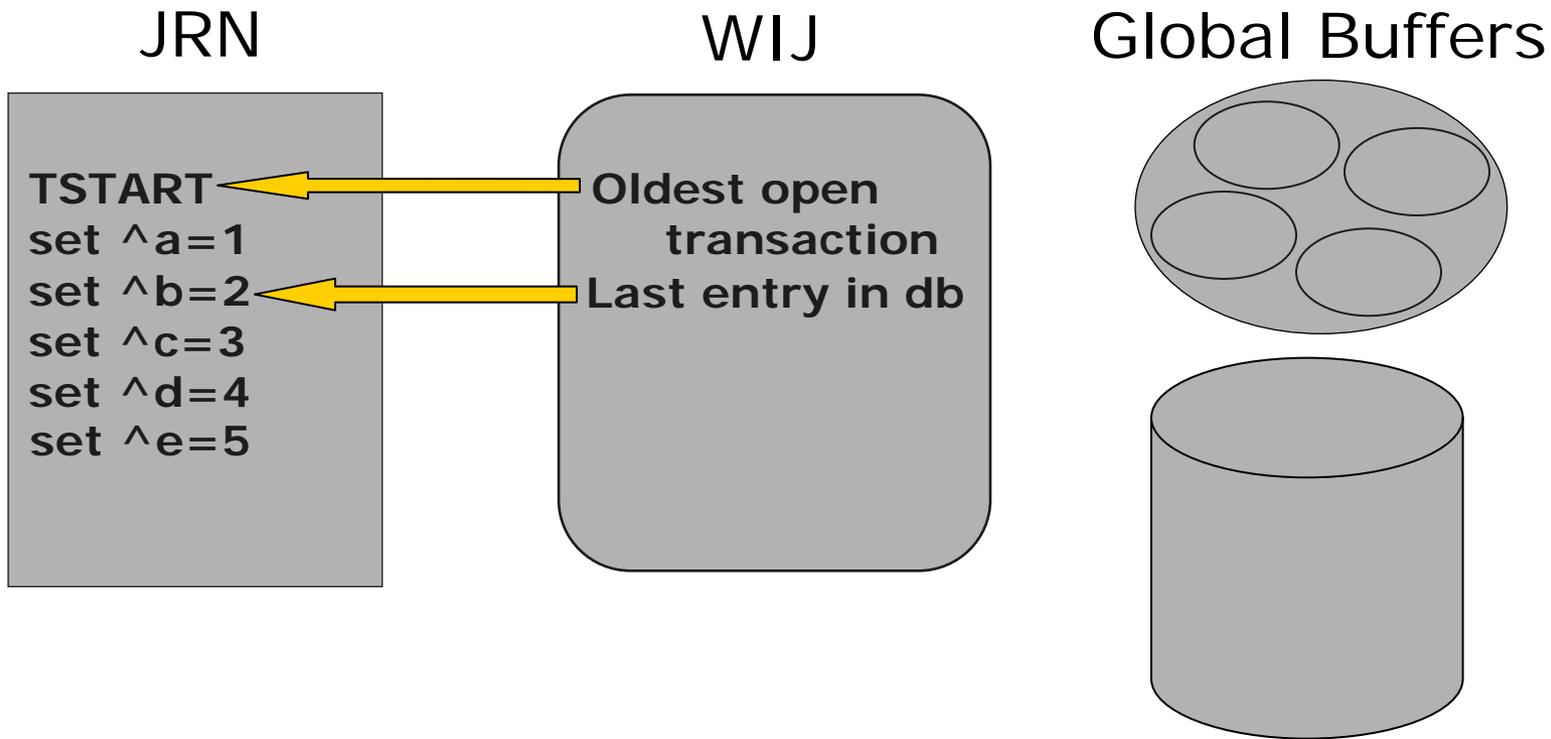
- Besides blocks being written to disk, WIJ contains:
  - Pointer to last journal entry successfully written to database.
  - Address of oldest open transaction.





# Crash Resiliency (cont.)

- Upon restart after crash, Caché automatically:
  - Rolls forward WIJ.
  - Rolls forward journal from pointer in WIJ.
  - Rolls back all open transactions from pointer in WIJ.
- Database never left in inconsistent state.





# Cache System Logs

- Several Cache system logs assist in diagnosing problems.
- Use *Home* → *System Logs*.
  - Shadowing and Backup pages also have access to logs.

## SYSTEM LOGS

View activity and error logs

- [View Application Error Log](#)
- [View Console Log](#)
- [View ODBC Error Log](#)



# Suggested Readings

---

- Caché High Availability Guide.
  - *Write Image Journaling and Recovery.*
  - *Journaling.*
- Using Caché ObjectScript.
  - *Transaction Processing.*



# Practice: Discuss Write Process



## ■ Example:

```
set br = ##class(FCE.Branch).%OpenId(IdA)
set br.Phone = "+1 617 720 1498"
set br.ATMs.GetAt(1).SerialNumber =
br.ATMs.GetAt(1).SerialNumber _ "TXX"
do br.%Save()
```

## ■ Discuss with your neighbour, which information is written to WIJ, Journal, and DB buffers

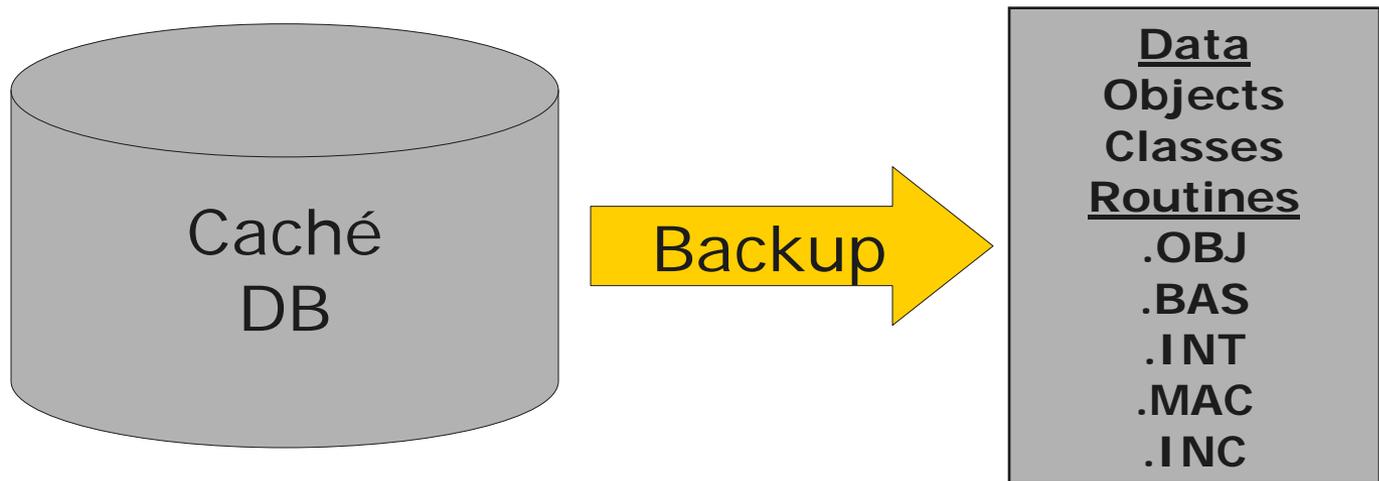
## ■ Note

- %Save() saves branch and affiliated ATM
- %Save() automatically performs both operations in a transaction



## 2.4 Caché Backup

- Caché allows backups while applications are running and while database is changing.
  - Referred to as “concurrent” backup.
  - Uses multiple passes to back up.





# Backup Passes

---

- Multiple passes allow for data integrity and snapshot of database in time.
- First pass
  - Begin tracking any blocks changing from this point.
    - Sets back-up bitmaps appropriately.
  - Backup all blocks.
- Second...nth pass
  - Backup any blocks changed during previous pass.
- Last pass
  - Write Daemon suspends during final pass—users continue uninterrupted.



# Types of Backup

---

- Full
  - Complete backup of database.
- Cumulative
  - Tracks all changes since last full backup.
- Incremental
  - Tracks all changes since last backup, regardless of previous backup type.
- Cold backup
  - Stop Caché
  - Copy CACHE.DAT files



# How To: Configure Backup

- Create Database List.
  - *Configuration → Database Backup Settings → Define Database Backup List.*
- Configure location for backup file.
  - *Configuration → Database Backup Settings → Choose appropriate type of backup.*

| DATABASE BACKUP SETTINGS           |                                                   |
|------------------------------------|---------------------------------------------------|
| Configure database backup settings |                                                   |
| ➤                                  | Define Database Backup List                       |
| ➤                                  | Configure Full Backup of All Databases            |
| ➤                                  | Configure Full Backup of the Database List        |
| ➤                                  | Configure Incremental Backup of the Database List |
| ➤                                  | Configure Cumulative Backup of the Database List  |



# How To: Run a Backup

- *Backup* → Choose appropriate type of backup.
- or do ^BACKUP.

Run Backup Task

**Name:** FullDBList  
**Description:** Full backup of all databases that are in the backup database list.  
**Type:** Full  
**Log File:** c:\cache825min\mgr\Backup\FullDBList\_20051212\_002.log  
**Save to Tape:**  Tape Number: 47  
**Device to save backup:** C:\Cache825Min\Mgr\Backup\

---

**The Backup Database List:** 1. TEST

---

To start this backup, click the OK button



# How To: Review Backup Results



- Management Portal.
  - While running:
    - *Backup* → Run Backup → *Click here to view status.*
  - After backup:
    - *Backup* → *View Backup History* → Click *View* for backup.
  - Log is created during backup.



# Restoring from Caché Backup

---

- No users should be on system.
- Restore last full backup of each database that is no longer accessible.
- Restore last cumulative backup since full backup.
- Restore all incremental backups since full backup or last cumulative backup in the order in which the backups were performed.
  - Perform database and application integrity checks.
- Apply changes in journal file for restored databases.
  - If restored ALL databases on system, can remove WIJ before startup.
- Perform full backup of restored system.



# Caché Restore Process

---

- To begin, restore process will prompt for:
  - Whether to suspend Caché processes while restoring takes place.
    - Recommended.
  - First file from which to restore.
- When restoring database have options:
  - Restore database to another location.
  - Restore only some globals in database.
- After first file is restored, restore process will prompt for:
  - Input file for next backup to restore.
    - Uses backup history to suggest next logical backup file.



## Caché Restore Process (cont.)

---

- After all backup files are restored, restore process will prompt for:
  - Which databases to restore journal files to.
  - Name of first journal file to restore.
    - Can choose to restore only some globals from journal file.
  - Name of last journal file to restore.



# Good Backup Practices

---

- Clearly mark backups.
- Define backup strategy, such as:
  - Keep daily backups for two weeks.
  - Keep weekly backups for two months.
  - Keep monthly backups for six months.
  - Keep quarterly backups for nine months.
- Test restores on regular basis.
  - Restore full, incremental and/or cumulative to alternate server.
  - Run integrity check.
- Exchange tapes after defined usage time.
- For Disaster Recovery, store backups in secure off-site location.



# Suggested Readings

- **Caché High Availability Guide.**
  - *Backup and Restore.*
  - *System Failover Strategies.*

Caché High Availability Guide

## Backup and Restore

User: **Unknow**

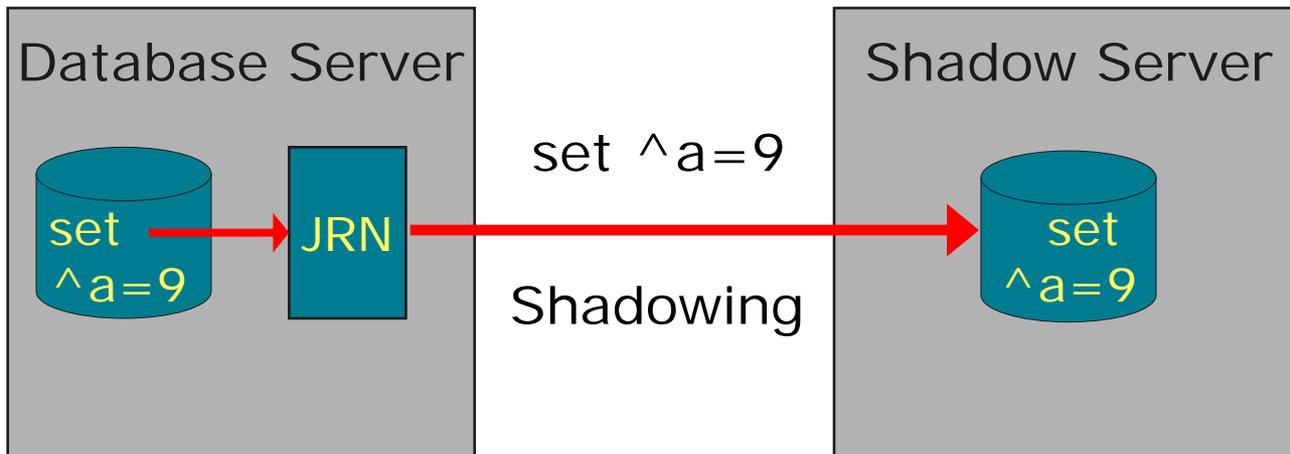
This chapter outlines the factors in developing a solid plan for backing up your Caché system. It discusses techniques for ensuring the integrity and recoverability of your backups, as well as suggested backup methodologies. Later sections of the chapter contain details about the procedures used to perform these tasks, either through the System Management Portal or by using Caché and third-party utilities. It discusses the following topics:

- [Backup Integrity and Recoverability](#)
- [Importance of Journals](#)
- [Backup Methods](#)
- [Configuring Caché Backup Settings](#)
- [Managing Caché Online Backups](#)
- [Restoring from a Backup](#)
- [Caché Backup Utilities](#)
- [UNIX Backup and Restore](#)
- [OpenVMS Backup and Restore](#)
- [Sample Backup Scripts](#)



## 2.5 Shadowing Operation

- Shadow server continually reads journal of DB server.
- Updates shadow server's database.
- Only shadow what is being journaled.
- Requires multi-server license.
- Shadow Daemons
  - Server-side process: SHDWSBLK
  - Client-side process: SHDWCBLK





# Shadowing Uses

---

- Run reports and queries on shadow server.
  - Can shadow multiple databases into one for reporting.
- Use shadow server as part of disaster recovery.
- Perform database backups on shadow server.



# How To: Setting Up Shadowing

- Configure shadowing on:
  - Database server machine.
  - Shadow machine.
- For database server:
  - Enable Journaling for database being shadowed.
  - Enable %Service\_Shadow.
  - Define Allowed Incoming Connections.
    - If not restricted, any machine that can see the database server can shadow it.
    - Enter IP addresses of shadow servers allowed to shadow this database.

Edit Service %Service\_Shadow

Service Name\*: %Service\_Shadow  
Description: Controls Shadow Clients

Service enabled:

Save Cancel

**Allowed Incoming Connections:**

| Index | IP Address   | Roles |        |
|-------|--------------|-------|--------|
| 1     | 198.119.22.3 | -     | Delete |

Add



# How To: Set Up Shadow Machine

- Verify TCP connection exists between shadow system and database server system.
- Configure shadow server settings. (Required)
  - *Configuration* → *Shadow Server Settings*.
    - Need to select journal to start shadowing.

|                                                                        |   |                                                      |
|------------------------------------------------------------------------|---|------------------------------------------------------|
| Name of the shadow:                                                    | 1 | TestShad2                                            |
| DNS name or IP address of the source:                                  | 2 | <input type="text" value="198.119.22.3"/>            |
| Port number of the source:                                             | 3 | <input type="text" value="1975"/>                    |
| Source event to start shadowing:                                       |   | 2005-12-12 11:15:14                                  |
| Journal switched to c:\cache825min\mgr\journal\20051212.002: <STARTUP> |   |                                                      |
| The Shadow service is Enabled on the source.                           |   | <input type="button" value="Select Source Event"/> 4 |



## How To: Set Up Shadow Machine (cont.)

- Set up directory mapping. (Required)
  - *Configuration* → *Shadow Server Settings* → *Add*.
  - Select source database directory.
    - Directory where server database resides.
  - Select shadow database directory.
    - Directory where shadow database resides.

**Source database directory:**

c:\cache\customer\

**Shadow database directory:**

C:\cache\test\

Save Cancel



# How To: Set Up Shadow Machine (cont.)

- Configure advanced settings. (Optional)
  - *Configuration* → *Shadow Server Settings* → *Add* → *Advanced*.

Location for copy of  
database server journal file  
on shadow machine.

|                                |                                                        |
|--------------------------------|--------------------------------------------------------|
| <b>Journal file directory:</b> | <input type="text" value="c:\cache\journals\shadow\"/> |
| <b>Filter routine:</b>         | <input type="text"/>                                   |
| <b>Maximum error messages:</b> | <input type="text" value="10"/>                        |

Routine to filter records  
before dejournaling.

# of shadow errors to retain.  
200 is maximum.



# How To: Review Shadow/Source Servers

**[ Home ] > [ Shadow Servers ]**

## **SHADOW**

Details for Shadow Server

➤ **This System as Shadow Server**

## **DATA SOURCE**

Details for Data Source

➤ **This System as Data Source**

➤ **Error Log**



# Disaster Recovery

---

- Can use shadow DB as master DB for disaster recovery.
  - Turn off shadowing on startup for shadow server.
  - Stop shadowing.
  - Change IP address and/or fully qualified domain name of shadow server to match that of database server.
  - Have users log in to applications.
- When shadow stopped, automatically rollbacks open transactions.
  - Guarantees transactional integrity.



# Suggested Readings

- Caché High Availability Guide.
  - *Shadow Journaling.*
- Technical Articles.
  - *CHUI-Based Management Routines.*

## Chapter 4: [Shadow Journaling](#)

- [4.1 Shadow Journaling Overview](#)
- [4.2 Enabling Shadow Journaling](#)
  - [4.2.1 Configuring the Source Database Server](#)
  - [4.2.2 Configuring the Destination Shadow](#)
  - [4.2.3 Setting Shadowing Properties](#)
- [4.3 Using Shadow Journaling](#)
  - [4.3.1 Starting Shadow Journaling](#)
  - [4.3.2 Stopping Shadow Journaling](#)
  - [4.3.3 Purging Shadow Journal Files](#)
  - [4.3.4 Viewing Error Messages](#)
- [4.4 Using the Shadow Destination for Disaster Recovery](#)



## 3. Security in Databases

---

- Databases often store most valuable information of a company or an organization.
- Data must be secured against unauthorized access
- Different security levels possible
  - Authentication: Only authorized users may access data
  - Authorization: Once logged in, different users have different access rights.
  - Data encryption: Even if a non-authorized person manages to access data, he cannot interpret them
  - Auditing: After data have been manipulated find out who did it, and when.



## 3.1 Authentication Definition

- Process of determining whether someone or something is, in fact, who or what it declares to be.
  - Check identity is valid.





# Caché Authentication

---

- Possible mechanisms:
  - Kerberos.
    - Secure authentication method over unsecured network.
  - OS-based.
    - OS user identity used to identify user to Caché.
  - Caché login.
    - Caché username/password account.
  - Unauthenticated.
    - Appropriate if organization has strongly protected perimeter, or data/application not attractive target.



# Authentication and Services

- Authentication is not set for Caché in total but for each connection service

| Service Name         | Kerberos Cache | Kerberos Login | OS-based | Caché Login | Unauthenticated |
|----------------------|----------------|----------------|----------|-------------|-----------------|
| %Service_Bindings    | N              | Y              | N        | Y           | Y               |
| %Service_CSP         | N              | Y              | N        | Y           | Y               |
| %Service_CacheDirect | N              | Y              | N        | Y           | Y               |
| %Service_CallIn      | Y              | Y              | Y        | Y           | Y               |
| %Service_ComPort     | N              | N              | N        | Y           | Y               |
| %Service_Console     | Y              | Y              | Y        | Y           | Y               |
| %Service_LAT         | N              | N              | N        | Y           | Y               |
| %Service_Telnet      | N              | Y              | N        | Y           | Y               |
| %Service_Terminal    | Y              | Y              | Y        | Y           | Y               |



# Demonstration



- Test authentication methods using Terminal
  - Unauthenticated.
  - Operating System.
  - Password.

```
Cache TRM:2416 (CACHE52)
File Edit Help
Username: mhulin
Password: *****
USER>
```



# OS-Based Authentication

---

- Available with connection services of:
  - Terminal (Local connections on UNIX/OpenVMS).
  - Console (Local connections on Windows).
  - CallIn (All platforms).
    - Only when directly invoking calls from OS-level prompt, not when using CallIn programmatically.
- Available only for local processes.



# Caché Login Authentication

- Uses hashed table to store passwords.
- Default password criteria:
  - 3.32ANP (Minimal and Normal security level).
  - 8.32ANP (Locked Down security level).

[Home] > [Security Management] > [System Security Settings]

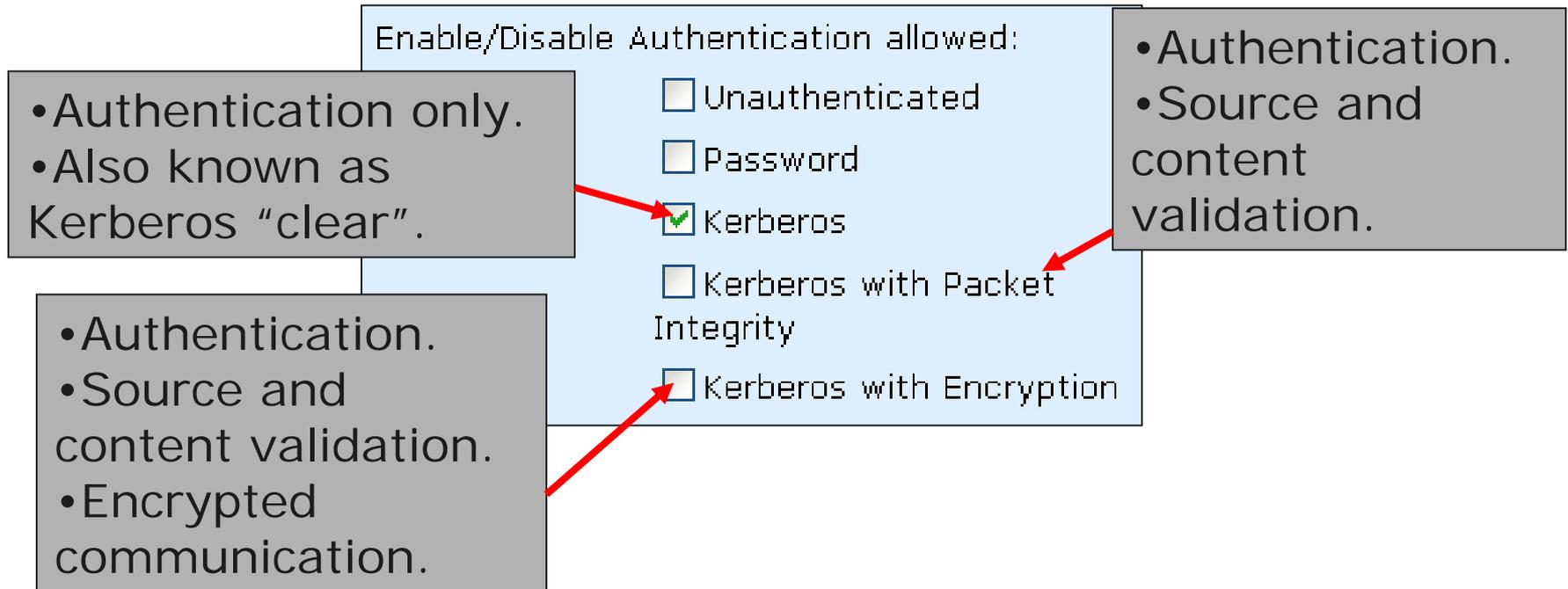
### Edit System-wide Security Parameters

|                                    |                                     |
|------------------------------------|-------------------------------------|
| Enable Audit:                      | <input checked="" type="checkbox"/> |
| Enable configuration security:     | <input type="checkbox"/>            |
| Default security domain:           | ISC                                 |
| Inactive limit (0-365):            | 90                                  |
| Invalid login limit (0-64):        | 5                                   |
| Password pattern:                  | 3.32ANP                             |
| Enable writing to percent globals: | <input type="checkbox"/>            |
| Allow multiple security domains:   | <input type="checkbox"/>            |



# Kerberos Options (C/S and CSP)

- Used with client-server and CSP connections.
- Can choose validation and encryption protection features.





# Kerberos Options (Local)

- Used with %Service\_Console (Windows) and %Service\_Terminal (UNIX/OpenVMS).

Enable/Disable Authentication allowed:

- Unauthenticated
- Operating System
- Password
- Kerberos
- Kerberos Credentials Cache

• Prompts user for username/password.  
• Also known as "Kerberos Login" and "Kerberos Prompting".

Gets username/password from cache.



# Using Kerberos

- Requires significant learning and personnel investment.
  - Do not use unless Kerberos administrator is possible.
- Caché security domains correspond to Kerberos realms.

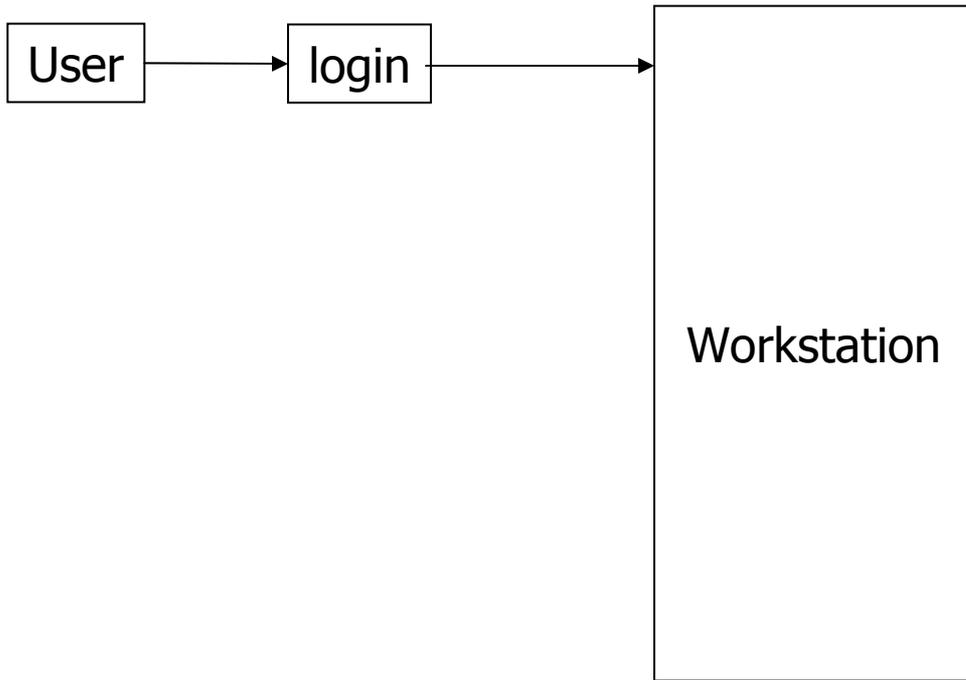
Enable/Disable Authentication allowed:

- Unauthenticated
- Password
- Kerberos
- Kerberos with Packet Integrity
- Kerberos with Encryption



# Operation of Kerberos

---





# Suggested Readings

- **Caché Security Administration Guide.**
  - *Authentication.*

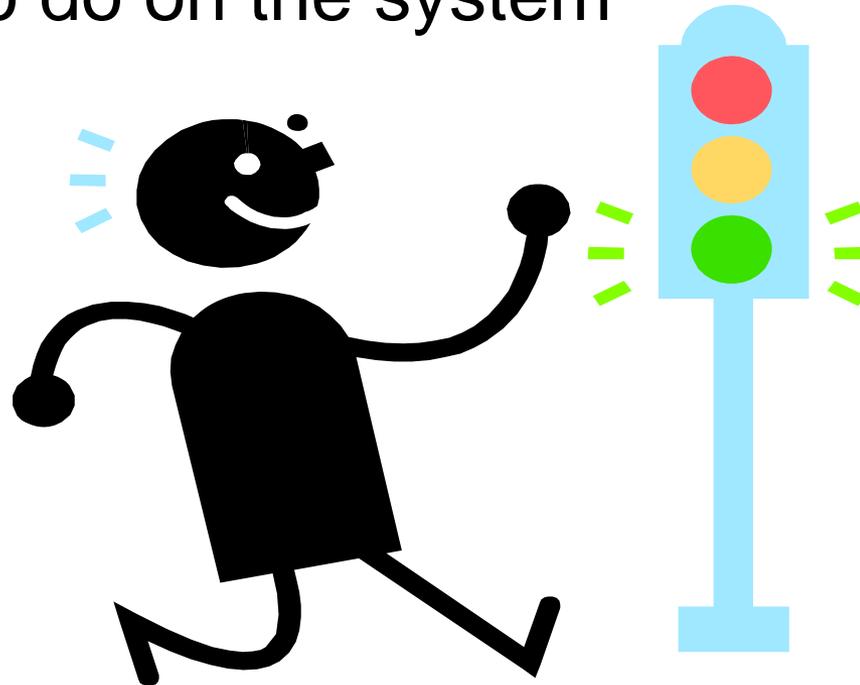
Chapter 2: **Authentication**

- **2.1 About the Different Authentication Mechanisms**
  - **2.1.1 Kerberos Authentication**
  - **2.1.2 Operating-System-Based Authentication**
  - **2.1.3 Caché Login**
- **2.2 About the Different Access Modes**
  - **2.2.1 About Local Access**
  - **2.2.2 About Client-Server Access**
  - **2.2.3 About CSP**
- **2.3 Configuring for Kerberos Authentication**
  - **2.3.1 About Kerberos and the Access Modes**
  - **2.3.2 Specifying Connection Security Levels**
  - **2.3.3 Setting Up a Client**
  - **2.3.4 Obtaining User Credentials**
  - **2.3.5 Setting Up a Secure Tunnel for a CSP Connection**
- **2.4 Configuring for Operating-System-Based Authentication**
  - **2.4.1 A Note on %Service\_Console**
  - **2.4.2 A Note on %Service\_Callin**
- **2.5 Configuring for Authentication with Caché Login**



## 3.2 Authorization Definition

- Process of giving someone permission to do or have something.
- Authorization defines **what** an authenticated user is allowed to do on the system

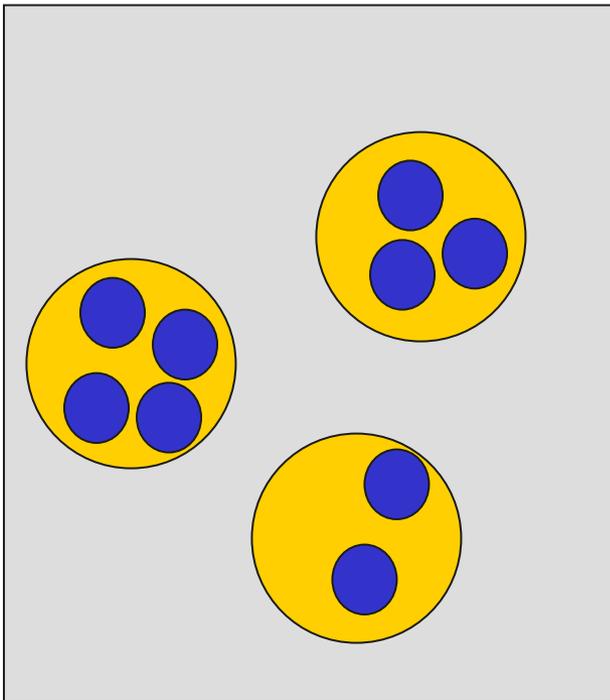




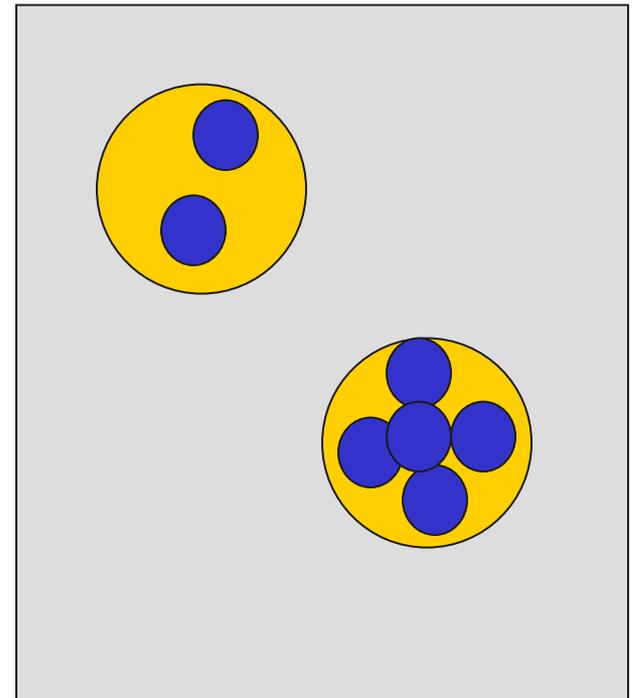
# Authorization Basics



User A



User B





# Users

- Define one Caché user for each person accessing Caché.

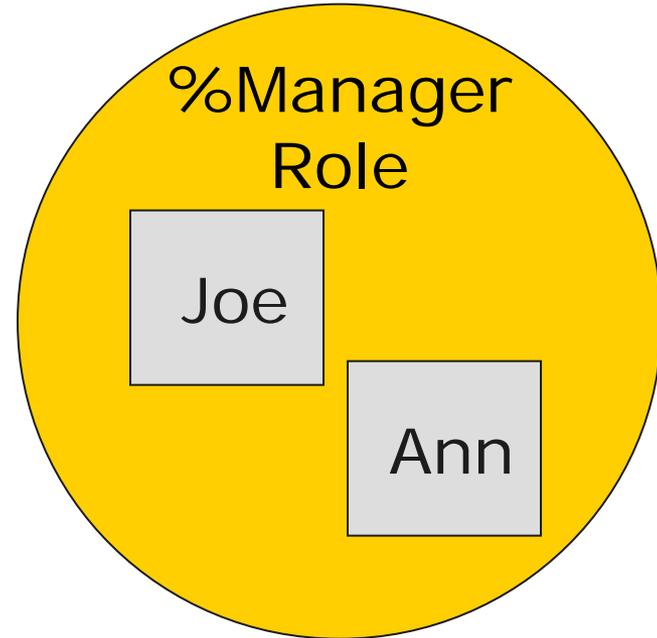
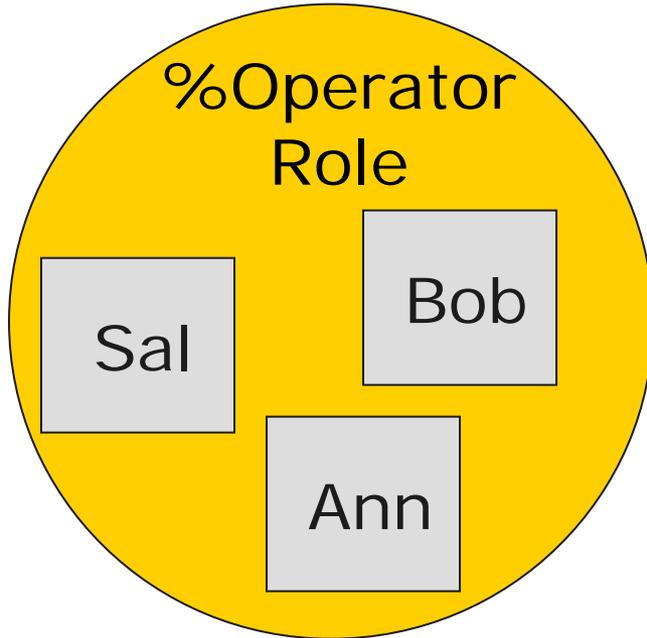
The following is a list of user definitions:

| User      | Description                    | Enabled | Namespace |
|-----------|--------------------------------|---------|-----------|
| _PUBLIC   | (Internal use - not for login) | No      | %SYS      |
| _SYSTEM   | SQL System Manager             | Yes     | %SYS      |
| Admin     | System Administrator           | Yes     | %SYS      |
| Chris     |                                | Yes     |           |
| CSPSystem | CSP Gateway user               | Yes     | %SYS      |
| Fran      |                                | Yes     |           |
| oleary    | User who installed system      | Yes     |           |
| SuperUser | System Super user              | Yes     | %SYS      |



# Roles

- Roles can have multiple members.
- Users can be members of multiple roles.





# Privileges

- Roles are named collections of privileges.
- Privileges provide *permissions* on resources.
  - Possible permissions: Read, Write or Use.
  - Possible resources: databases, services and others.

## Privileges of %Operator Role

| Resource       | Permission  |
|----------------|-------------|
| %DB_CACHETEMP  | Read, Write |
| %DB_DOCBOOK    | Read        |
| %Service_CSP   | Use         |
| %Admin_Operate | Use         |



# Authorization Basics Overview

- User can have permissions from multiple roles.

User has assigned roles.

↳ Role has assigned privileges.

↳ Privilege grants permission.

| User | Roles     | Privileges     |             |
|------|-----------|----------------|-------------|
|      |           | Resource       | Permission  |
| A    | %Operator | %DB_CACHETEMP  | Read, Write |
|      |           | %DB_DOCBOOK    | Read        |
|      |           | %Service_CSP   | Use         |
|      |           | %Admin_Operate | Use         |
|      | %DB_USER  | %DB_USER       | Read, Write |
|      | %SQL      | %Service_SQL   | Use         |



# Predefined Users

| Username    | Assigned Roles                 | Purpose                                                         |
|-------------|--------------------------------|-----------------------------------------------------------------|
| Admin       | %Manager                       | Admin account.                                                  |
| CSPSystem   | %All                           | Login for CSP gateway for Normal and Locked Down installations. |
| SuperUser   | %All                           | Account with all privileges available.                          |
| UnknownUser | %All – Minimal (None) – Others | Account for unidentified user.                                  |
| _PUBLIC     | (None)                         | Set of privileges given to all users (not a login account).     |
| _SYSTEM     | %ALL                           | Account to support SQL access.                                  |



## Predefined Users (cont.)

---

- SuperUser
  - Exists to help with initial security configuration.
  - Remove after initial set up is complete.
- `_PUBLIC`
  - Used to assign roles to all users connected to the system.
    - Add roles as desired.
- `_SYSTEM`
  - Supports standard SQL specification.



# UnknownUser

- Used when unauthenticated user connects to system.
  - Service must allow unauthenticated access.
  - Not used when authentication fails.
- Unknown user has privileges from:
  - Public resources.
  - Roles of UnknownUser.
  - Roles of \_PUBLIC.
- In minimal install, all users connect as UnknownUser (has %All role).

|                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------|
| Service Name*: %Service_CSP<br>Description: Controls CSP application pages<br>Service enabled: <input checked="" type="checkbox"/> |
| Enable/Disable Authentication allowed:<br><input checked="" type="checkbox"/> Unauthenticated                                      |

|                                      |
|--------------------------------------|
| User Name: <input type="text"/>      |
| Password: <input type="password"/>   |
| <input type="button" value="Login"/> |



# Default User Passwords

Change existing default passwords.

| Installation Type | Default Password            | Exceptions                                        |
|-------------------|-----------------------------|---------------------------------------------------|
| Minimal           | SYS                         | _PUBLIC has no password                           |
| Normal            | Defined during installation | _PUBLIC has no password                           |
| Locked Down       | Defined during installation | _PUBLIC has no password<br>_SYSTEM is not enabled |



# Deleting Predefined Users

---

- Can delete any unwanted predefined user account.
  - But, at least one user must have %All role.
    - Does not matter which user has %All.
  - System will not allow deletion of last user with %All role.



# How To: Create Users

[Home] > [Security Management] > [Users] > [Edit User]

Create User Definition

[General]

Roles

SQL Privileges

SQL Tables

SQL Views

SQL Procedures

|                                                                          |                                                                                     |
|--------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Name*:                                                                   | <input type="text"/>                                                                |
| Copy from:                                                               | <input type="text" value="an existing name here"/> <input type="button" value="v"/> |
| Full Name:                                                               | <input type="text"/>                                                                |
| Comment:                                                                 | <input type="text"/>                                                                |
| Password*:                                                               | <input type="password"/>                                                            |
| Confirm Password*:                                                       | <input type="password"/>                                                            |
| User Enabled:                                                            | <input checked="" type="checkbox"/>                                                 |
| Expiration Date:                                                         | <input type="text"/> (yyyy-mm-dd)                                                   |
| Default Namespace:                                                       | <input type="text"/> <input type="button" value="v"/>                               |
| Default Tag^Routine:                                                     | <input type="text"/>                                                                |
| <input type="button" value="Save"/> <input type="button" value="Close"/> |                                                                                     |



# Predefined Roles

| Role           | Description                    |
|----------------|--------------------------------|
| %All           | The Super-User Role            |
| %DB_%DEFAULT   | R/W access for this resource   |
| %DB_CACHEAUDIT | R/W access for this resource   |
| %DB_CACHELIB   | R/W access for this resource   |
| %DB_CACHESYS   | R/W access for this resource   |
| %DB_CACHETEMP  | R/W access for this resource   |
| %DB_DOCBOOK    | R/W access for this resource   |
| %DB_SAMPLES    | R/W access for this resource   |
| %DB_USER       | R/W access for this resource   |
| %Developer     | A Role owned by all Developers |
| %Manager       | A role for all System Managers |
| %Operator      | System Operators               |
| %SQL           | Role for SQL access            |



## Predefined Roles (cont.)

---

- All predefined roles are optional, except %All.
  - Can delete predefined roles.
- %All
  - Cannot be deleted or modified.
  - At least one user account must have %All role.



# How To: Create Roles

- Role naming conventions:
  - Must not start with %.
  - Cannot differ from another role name only by case.

[Home] > [Security Management] > [Roles] > [Edit Role]

Edit Role Definition **TestRole**

[General] [Members] [SQL Privileges] [SQL Tables] [SQL Views] [SQL Procedures]

Name\*: TestRole

Description:

Privileges:

| Resource     | Permission |      |        |
|--------------|------------|------|--------|
| %DB_USER     | RW         | Edit | Delete |
| %Service_CSP | U          | Edit | Delete |

Add...

Save Close



# How To: Create Roles (cont.)

Edit definition for role TeamMember

**General** **[Members]** SQL Privileges SQL Tables SQL Views SQL

| Members of this role       | Member Type | Option                 |
|----------------------------|-------------|------------------------|
| <a href="#">Jack</a>       | USER        | <a href="#">Remove</a> |
| <a href="#">Supervisor</a> | ROLE        | <a href="#">Remove</a> |

[Remove All](#)

Possible member types:  
- Users  
- Roles

Select Option:

**Grant users to this role**  Grant other roles to this role  Grant this role to other roles

Hold the [Shift] or [Ctrl] key while clicking to select multiple.

**Available** **Selected**

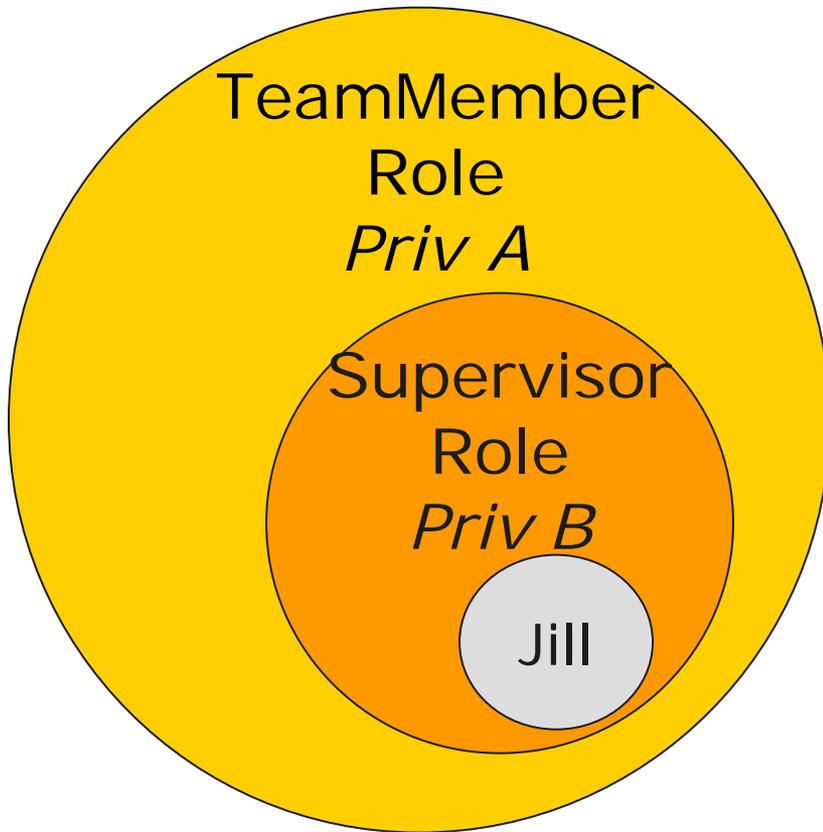
— Select One or More —

Admin  
CSPSystem  
SuperUser

[Assign](#) [Assign with Grant](#)



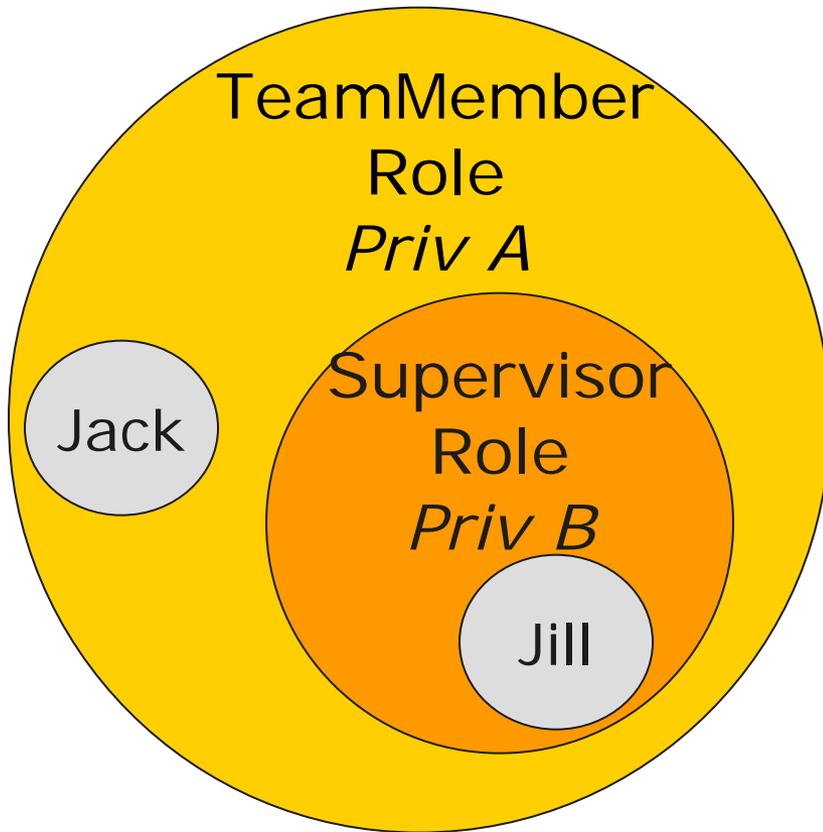
# Role as Role Member



- Jill is a Supervisor.
- Supervisor is a TeamMember.
- Jill has following privileges:
  - Priv A.
  - Priv B.



## Roles as Role Members (cont.)



- Jack is a TeamMember.
- Jack has Priv A.
- But, Jill has Priv A and Priv B.



# Roles as Role Members (cont.)

User **Jill** holds the following privileges:

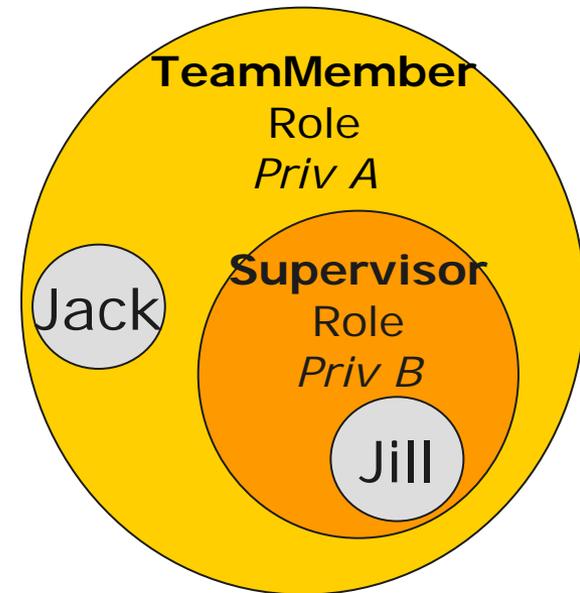
| Asset          |                                                       | Privileges |   |   | Granted by role |
|----------------|-------------------------------------------------------|------------|---|---|-----------------|
| Resource       | Protects                                              | R          | W | U |                 |
| %Admin_Manage  | System manager tasks<br>Applications:<br>/csp/sys/mgr |            |   | U | Supervisor:U    |
| %Admin_Operate | System operator tasks<br>Applications:<br>/csp/sys/op |            |   | U | TeamMember:U    |

Edit definition for role **TeamMember**

General [Members] SQL Privileges SQL

| Members of this role | Member Type |
|----------------------|-------------|
| <u>Jack</u>          | USER        |
| <u>Supervisor</u>    | ROLE        |

Remove All





# Privileges

**Privilege = Resource:Permission**

Example: Privileges for %Operator Role

%Admin\_Operate:Use

%DB\_CACHETEMP:Read, Write

%DB\_DOCBOOK:Read

%Service\_CSP:Use



# Resources

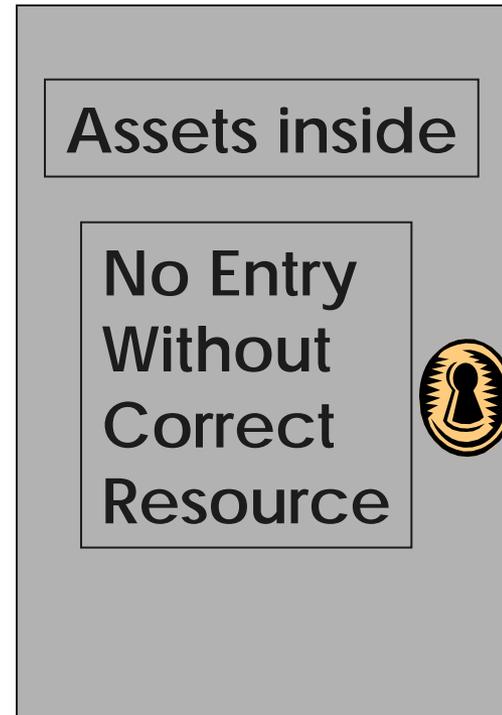
- After user has authenticated, resources control whether assets are accessible to user.
  - Resource with proper permissions are required to access assets.



**Resource A**



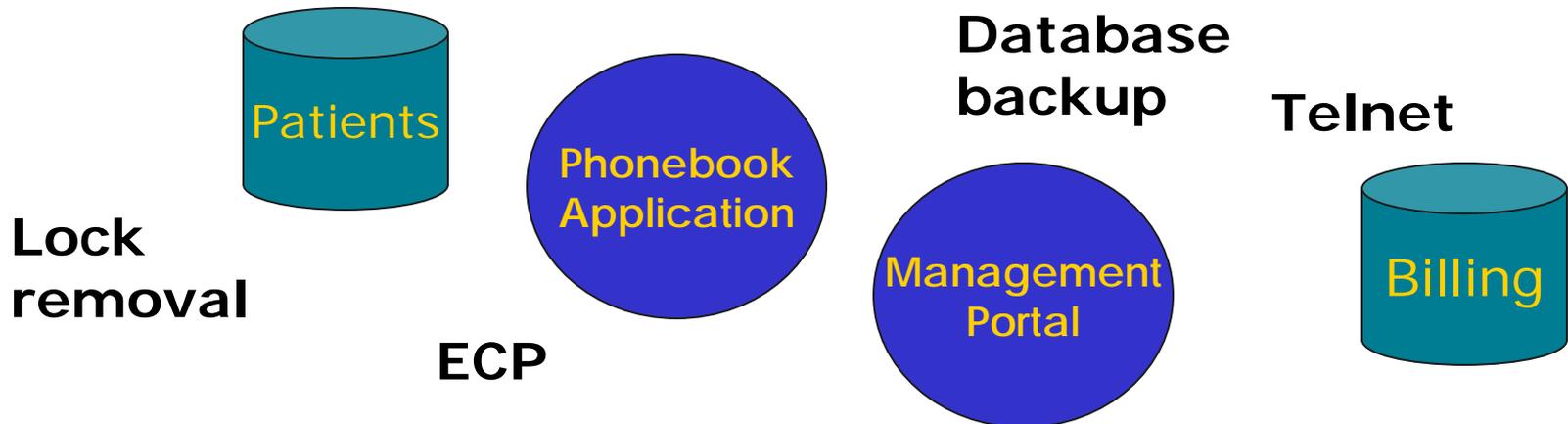
**Resource B**





# Assets

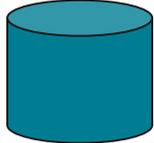
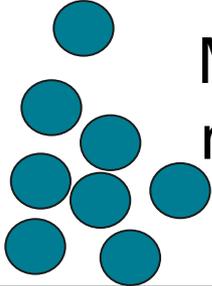
- Something that is protected.
- Examples include:
  - Caché database.
  - Caché services.
  - Applications.
  - System management functions.





# Assets and Resources

- Below are sample relationships between assets and resources.
  - One resource could be used to protect multiple databases.

| Resource                                                                                          | Asset(s) Protected                                                                                                   | Relationship |
|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|--------------|
|  %DB_CACHESYS    |  CACHESYS                          | One-to-one   |
|  %Admin_Manage |  Many system management functions | One-to-many  |



# Resource Types

| Type                                                                                             | Name            | Permission   |
|--------------------------------------------------------------------------------------------------|-----------------|--------------|
| Administration  | %Admin_<...>    | Use          |
| Database        | %DB_<...>       | Read, Write  |
| Development     | %Development    | Use          |
| Service         | %Service_<...>  | Use          |
| System        | %System_CallOut | Use          |
| User-defined  | User-defined    | User-defined |



# Predefined Resources

- Start with %.
- For resource descriptions:
  - Refer to *Technical Articles* → *Frequently Asked Questions About Caché Advanced Security*.

Examples

**%Development**

**%DB\_%DEFAULT**

**%Service\_Telnet**

**%Admin\_Manage**

**%DB\_CACHEAUDIT**

**%Service\_SQL**



# Public Resource

Resource Name\*: %DB\_DOCBOOK

Description:

Public Permission:  Read  Write

- Any resource can be designated as public.
- Equivalent to all users holding privilege for resource.



# Demonstration Authorization



- Look at predefined users.
- Create new user.
- Create new role.
- Look at predefined resources.
- Define databases with according resources
- Assign resources to DemoRole
- Assign users to DemoRole

**SECURITY DEFINITIONS**  
Configure security definitions for this system

- Users
- Roles
- Services
- Resources
- Auditing
- Security Advisor
- Security Domains
- System Security Settings



# Suggested Readings

---

- **Caché Security Administration Guide.**
  - *Introduction.*
  - *About Authorization.*
  - *Assets and Resources.*
  - *Privileges and Permissions.*
  - *Users.*
  - *Roles.*
  - *Services.*
- **Technical Articles.**
  - *Frequently Asked Questions About Caché Advanced Security.*



## 3.3 Database Encryption

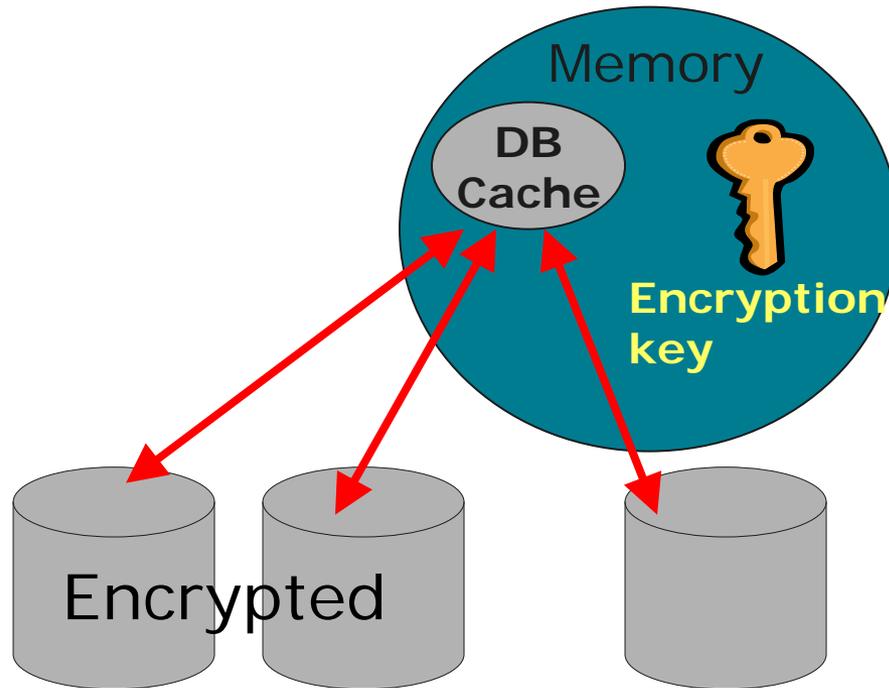
---

- Caché DB encryption protects data stored on disk.
- Caché uses the AES (Advanced Encryption Standard) algorithm with 128-bit keys by default.
- Encryption and decryption occur when Caché writes to or reads 8k-blocks from disk.
- Caché encrypts data, indices, bitmaps, pointers, allocation maps, and incremental backup maps.
- Encryption doesn't slow down DB access:
  - A typical block read lasts 8 ms on a 1 GHz-Computer
  - Encryption adds only 0,24 ms



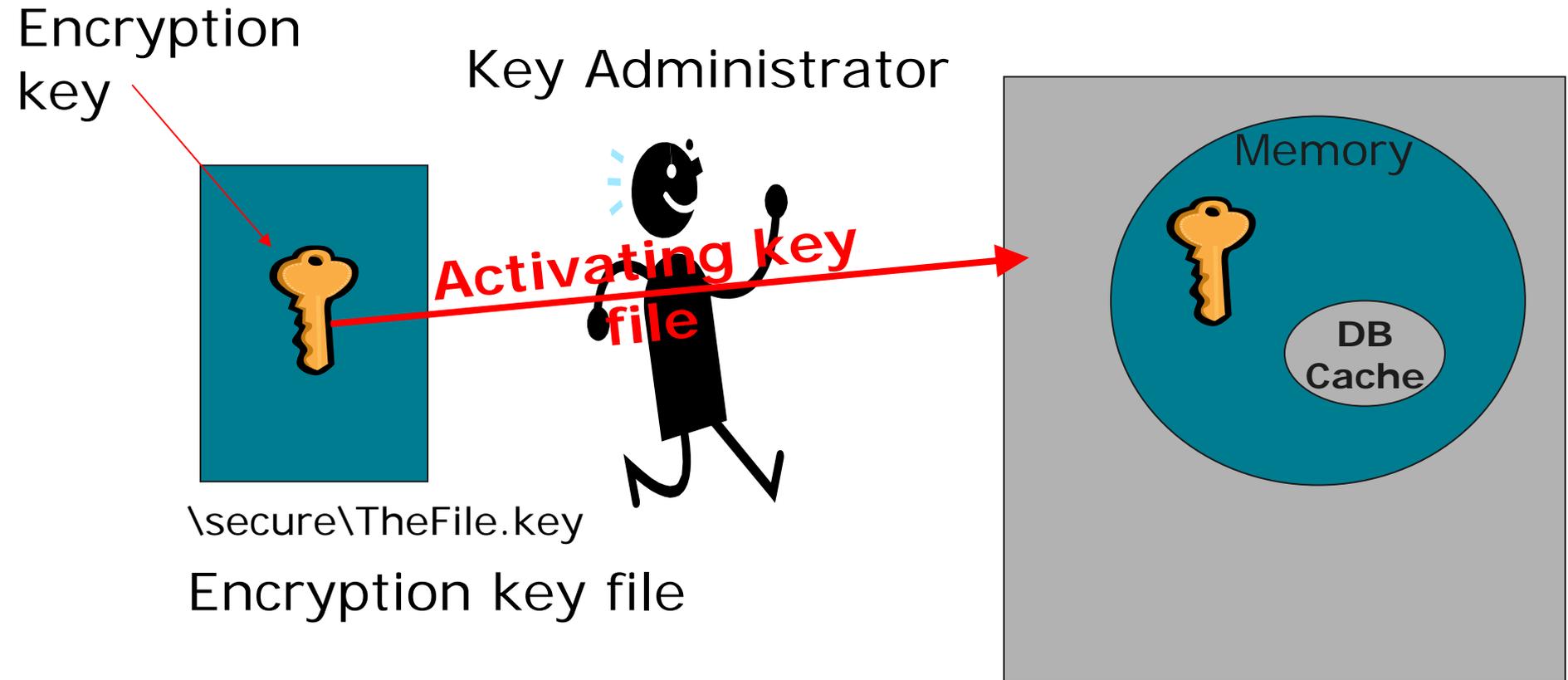
# Encryption

- Protects data at rest.
  - Allows secure on-disk storage of CACHE.DAT files.





# Encryption Terms





# How To: Create Encryption Key File

[Home] > [Database Encryption] > [Create Encryption Key]

|                                    |                      |                                          |
|------------------------------------|----------------------|------------------------------------------|
| Encryption Key Path and File Name: | <input type="text"/> | <input type="button" value="Browse..."/> |
| Administrator Name:                | <input type="text"/> |                                          |
| Password:                          | <input type="text"/> |                                          |
| Confirm Password:                  | <input type="text"/> |                                          |

- Administrator name and password need not exist in Caché.
  - Separate from Caché Advanced Security.
- Password should be random mixture of upper- and lower-case alphabetic characters, numerals, and punctuation.



# How To: Activate Encryption Key File

[Home] > [Database Encryption]



- Creating new key activates key automatically.



# How To: Manage Encryption Key File

**[ Home ] > [ Database Encryption ] > [ Manage Key File ]**

Encryption is activated.  
Encryption Key Identifier: 92720806-6041-4863-95F6-113EDC4383E2

Encryption Key Path and File Name:

**Administrators Defined in Key File [C:\Cache826min2\Mgr\EncrFile.key]**

| Count | Administrator |                        |
|-------|---------------|------------------------|
| 1     | NEWADMIN      | <a href="#">Delete</a> |
| 2     | FIRSTADMIN    | <a href="#">Delete</a> |

- Can add or remove administrators from encryption key.
- Remove any administrators who should no longer have access from all copies of all keys.



# Creating Encrypted Database

- Check *Encrypt Database?* when creating database.
  - Automatically creates and mounts with active key.
- Requires active encryption key.

**Database Wizard**

This wizard will help you create a new database.

Enter the name of your database: ENCRYP

Database directory: C:\Cache826min2\Mgr\ENCRYP

You may create an Encrypted Database if Encryption is activated.

**Encrypt Database?**



# Mounting Encrypted Database

---

- Requires active encryption key.
- To mount encrypted database on startup, need key active before databases are mounted.
- Two options for key activation at startup.
  - Interactive.
  - Unattended.



# Interactive Startup

- Prompts for key file, admin name and password at startup.
  - Most secure.
  - Requires direct admin involvement.
- *Database Encryption → Encryption Settings.*

Configure Caché startup and other options for encrypted databases.

## Startup Options

Activate database encryption key at Caché startup: **Interactive** ▼

## Other Options

Encrypt CacheTemp: **No** ▼ This will take effect upon next restart of the system.

Encrypt journal files: **No** ▼ This will take effect upon next restart of the system OR when the journal file is next switched (if encryption is enabled).

Save

Cancel

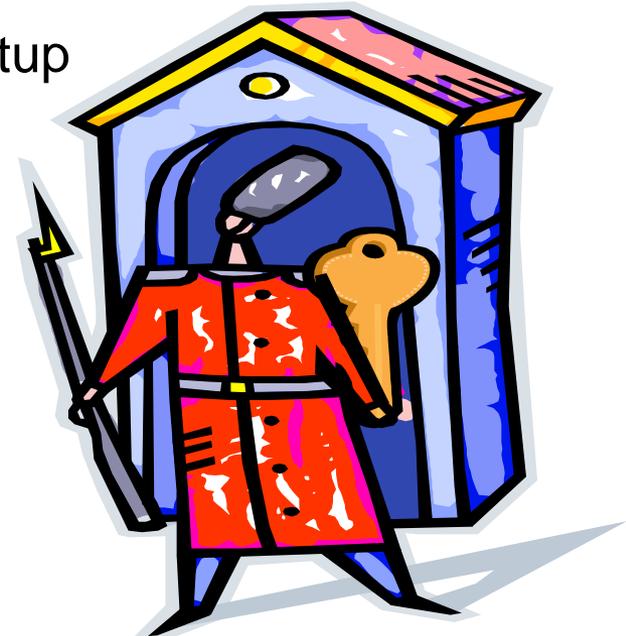


# Interactive Startup (cont.)

- Key file can be removed from system once activated.
- Quis custodiet ipsos custodes? A key keeper without username/password has key file under his control.
  - Can store key file in secure location and require admin to request access when needed.
  - Only hands key to key administrator for startup



Key administrator



Key keeper



# Startup Encryption Options

- Unattended and Interactive Startup allow specification of encrypting:
  - CACHETEMP.
  - Journal files.
- Only encrypt if necessary.

## Other Options

Encrypt CacheTemp:  This will take effect upon next restart of the system.

Encrypt journal files:  This will take effect upon next restart of the system OR when the journal file is next switched (if encryption is enabled).



# Unattended Startup

- Adds new "system" key-admin with long, random password to key file.
  - This admin used to activate key file automatically at Caché-startup.
  - Password stored in %SYS.
- Does not require a real person as key-admin.
- Does require copy of key on server.
  - Physical security is crucial. 
- *Database Encryption → Configure Startup.*

If an unauthorized intruder gets access to the DB-server he also has access to the key file.

Encryption Key Path and File Name: C:\CacheSys\Mgr\EncrpfFile.key

Save Cancel

**Administrators Defined in Key File [C:\Cache826r**

| Count | Administrator                        |        |
|-------|--------------------------------------|--------|
| 1     | NEWADMIN                             | Delete |
| 2     | FIRSTADMIN                           | Delete |
| 3     | 92720806-6041-4863-95F6-113EDC4383E2 | Delete |

Add



Added Admin



# Protecting Key File

---

- Always keep backup on non-volatile media in secure place.
- Keep activation copies on:
  - Non-removable, read-only media.
  - Or, on media with restricted access.
- Key-keeper should not be key admin.
- Backup key file should have one recoverable password.
  - For example, one locked-up password.



## Protecting Key File (cont.)

---

- Remove admin from key file when leaves, or no longer has need of key file access.
- Re-key databases if not sure all copies of keys are known when necessary to remove admin from key file.
  - Use *cvencrypt* executable program in \bin directory of Caché.



## Some Best Practices

---

- Physically secure servers.
- Restrict access to CACHE.DATs.
- When using ECP with secure databases, secure network traffic since ECP messages not encrypted.
- If your application audits changes to encrypted database, then encrypt CACHEAUDIT.
  - Apply best practices to audit database as well.



# Encryption and Other Data

---

- In WIJ, blocks for encrypted database are encrypted.
- Journal files can be encrypted as of v5.2.
- Always need encryption key activated if using encrypted journal files.
- Caché backup files are unencrypted.
  - OS backup of encrypted CACHE.DAT would be encrypted.
- Shared memory is unencrypted.
  - Core files have data from shared memory in unencrypted form.



## Suggested Readings

---

- Caché Security Administration Guide.
  - *Database Encryption.*
  - *Using the cvencrypt Utility.*



## 3.4 Auditing

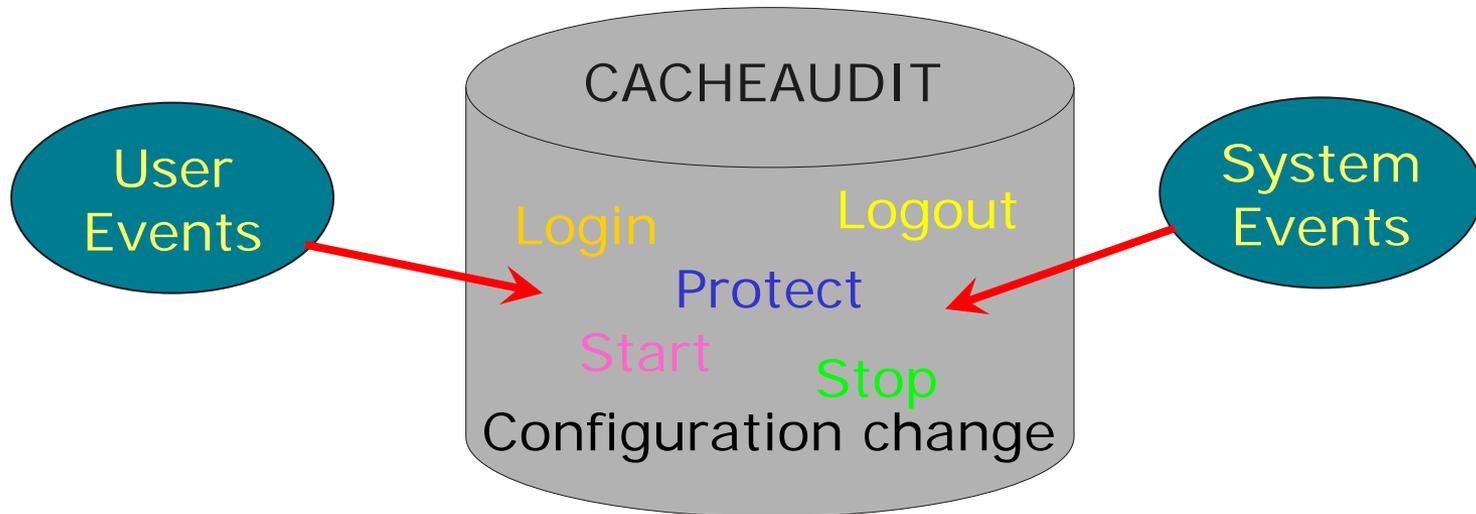
- Audit track: Record showing who has accessed the system and what operations they have performed during a given period of time.
- Auditing provides additional security: If you are suspicious that an unauthorized person accessed the database you can check the audit track.





# Auditing

- Logs certain key events in secure audit database.
  - Mandatory system events.
  - Optional system events.
  - User events.





# Auditing Basics

- Select events to audit.
- Enable auditing.
- System automatically creates entry in CACHEAUDIT database when events occurs.
- CACHEAUDIT is a standard Caché database.
- Review audit entries as needed.

| # | Time                    | Event Source | Event Type | Event       | PID  | GSP Session | User        | Description |
|---|-------------------------|--------------|------------|-------------|------|-------------|-------------|-------------|
| 1 | 2005-12-21 14:58:47.174 | %System      | %Security  | AuditChange | 4856 | 13deCsui00  | UnknownUser | Create Eve  |
| 2 | 2005-12-21 14:56:46.525 | %System      | %Security  | AuditChange | 4856 | 13deCsui00  | UnknownUser | Delete Eve  |
| 3 | 2005-12-21 14:56:32.071 | %System      | %Security  | AuditChange | 4856 | 13deCsui00  | UnknownUser | Create Eve  |
| 4 | 2005-12-21 14:42:06.737 | %System      | %Security  | AuditReport | 2840 | 13deCsui00  | UnknownUser | List query  |
| 5 | 2005-12-21 14:41:57.905 | %System      | %Security  | AuditReport | 2840 | 13deCsui00  | UnknownUser | List query  |
| 6 | 2005-12-21 14:41:56.420 | %System      | %Security  | AuditReport | 2840 | 13deCsui00  | UnknownUser | List query  |



# Mandatory System Events

- Logged whenever auditing is enabled.

%System/%Security/ApplicationChange

%System/%Security/AuditChange

%System/%Security/AuditReport

%System/%Security/DomainChange

%System/%Security/ResourceChange

%System/%Security/RoleChange

%System/%Security/ServiceChange

%System/%Security/SystemChange

%System/%Security/UserChange

%System/%System/AuditRecordLost

%System/%System/ConfigurationChange

%System/%System/Start

%System/%System/Stop

%System/%System/UserEventOverflow



# Optional System Events

- Logged when event is enabled.
  - DirectMode will log all Terminal session commands.

| Event Name                     | Enabled |
|--------------------------------|---------|
| %System/%DirectMode/DirectMode | No      |
| %System/%Login/Login           | No      |
| %System/%Login/LoginFailure    | No      |
| %System/%Login/Logout          | No      |
| %System/%Security/Protect      | No      |



# User Events

- User-defined application events.
  - Configure in Management Portal.
- Add audit entry by calling `$SYSTEM.Security.Audit()` in code.

| Event Name                      |
|---------------------------------|
| MyAppSource/MyAppType/MyAppName |

- If event occurs which is not predefined, it is written into audit log as:
  - Event = "UserEventOverflow".
  - Description = "Added unknown audit event".



# Demo



- Enable auditing.
- Create and view audit entries.
- Create user event to audit.

**SYSTEM AUDITING**  
Configure and use the system auditing database

- Enable Auditing
- ▶ Disable Auditing
- ▶ View Audit Database
- ▶ Configure System Events
- ▶ Configure User Events
- ▶ Manage Audit Log

Enable auditing

Enable events



# How To: View Audit Entries

- *Security Management* → *Auditing* → *View Audit Database*.
  - Can query based on multiple criteria.
- *^%AUDIT* → *Audit reports*.

| Event Source:                                                              | <input type="text" value="*"/>                   | <input type="button" value="..."/> | <table border="1"><thead><tr><th>Users</th></tr></thead><tbody><tr><td>* (All)</td></tr><tr><td>Admin</td></tr><tr><td>CSPSystem</td></tr><tr><td>Jack</td></tr><tr><td>Jill</td></tr><tr><td>SuperUser</td></tr><tr><td>UnknownUser</td></tr><tr><td>_PUBLIC</td></tr><tr><td>_SYSTEM</td></tr></tbody></table> | Users | * (All) | Admin | CSPSystem | Jack | Jill | SuperUser | UnknownUser | _PUBLIC | _SYSTEM |
|----------------------------------------------------------------------------|--------------------------------------------------|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------|-------|-----------|------|------|-----------|-------------|---------|---------|
| Users                                                                      |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| * (All)                                                                    |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Admin                                                                      |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| CSPSystem                                                                  |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Jack                                                                       |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Jill                                                                       |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| SuperUser                                                                  |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| UnknownUser                                                                |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| _PUBLIC                                                                    |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| _SYSTEM                                                                    |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Event Type:                                                                | <input type="text" value="%Security"/>           | <input type="button" value="..."/> |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Event Name:                                                                | <input type="text" value="*"/>                   | <input type="button" value="..."/> |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| System IDs:                                                                | <input type="text" value="*"/>                   |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Begin Date & Time:                                                         | <input type="text" value="2005-12-21 00:00:00"/> |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| End Date & Time:                                                           | <input type="text" value="2005-12-21 15:20:09"/> |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Sort by:                                                                   | <input type="text" value="Reverse Date"/>        | <input type="button" value="v"/>   |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Max Rows To Load:                                                          | <input type="text" value="1000"/>                | <input type="button" value="v"/>   |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| Color By:                                                                  | <input type="text" value=""/>                    | <input type="button" value="v"/>   |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |
| <input type="button" value="Search"/> <input type="button" value="Reset"/> |                                                  |                                    |                                                                                                                                                                                                                                                                                                                  |       |         |       |           |      |      |           |             |         |         |



# Suggested Readings

- **Caché Security Administration Guide.**
  - *Auditing.*

Chapter 11: **Auditing**

- **11.1 Basic Auditing Concepts**
  - **11.1.1 Enabling or Disabling Auditing**
- **11.2 About Audit Events**
  - **11.2.1 Elements of an Audit Event**
  - **11.2.2 About System Audit Events**
  - **11.2.3 Enabling and Disabling Optional System Events**
  - **11.2.4 About User Events**
- **11.3 Managing Auditing and the Audit Database**
  - **11.3.1 Viewing the Audit Database**
  - **11.3.2 Managing the Audit Database**
- **11.4 Other Issues**
  - **11.4.1 Freezing Caché If There Can Be No Audit Log Writes**
  - **11.4.2 About Counters**



# Practice: Discuss Methods to Secure Database



- Discuss with your neighbor, what a DB administrator can do to secure a database.
- Also think about methods not mentioned on the slides.

## **E-1: UML class diagram for the Car Rental System (CRS)**

Create a UML class diagram for the CRS Example together with your partner. One partner should design the part with Person, Address, PhoneNumber, Customer, Employee, and Branch. The other partner should create a UML class diagram for the part with Vehicle, Truck, Car, Motorcycle, Model, Manufacturer, and PaymentRate.

Note that a person's phone number should be defined as PhoneNumber type. However, make a branch's phone number a predefined type String for simplicity of the later exercise.

For a description of the sample scenario go back to lesson ["Sample Application"](#).

When both partners have completed their parts they should combine them and, together, add the Loan class. One of them should upload the solution as PDF file; the other only a text file with the comment: "Exercise completed together with ..."

Please remind me by a short e-mail to look at your solution and mark this exercise as "completed".

## **E-2: Create OOP unit test for CRS. pojos.Branch class. Create the CRS.pojos.Branch class.**

### **Things to know before you start all exercises hereafter**

- Some classes that are to be imported may have to be modified often according to more specifications and need in later exercises.
- ConnectDB class must be modified according to your Cache configuration.
- DBService class provides predefined methods that can be used any time at your convenience. Additional methods can be created to suit your needs. Some may have to be modified or uncommented once more classes are created or imported into the project.

### **These are possible specifications for objects of pojos.Branch class**

- A branch must have a name
- A branch must have a phone number
- The phone number must contain only numbers or the characters + or - or blanks. + can be in front only. - can not be in front.
- Phone number has to be unique
- The opening time can not be before 5:00 am
- The closing time can not be after 10:00 pm

### **Decide in your learning team which partner should design the test class for pojos.Branch (step 1-9) and which designs the Branch class itself (step 10-19)**

1. Add JUnit into the project's build path(Project > Properties > Java build path > Add Library > JUnit > JUnit4)
2. Import ConnectDB and DBService into the project under "connection" package. Adapt the connection parameters in ConnectDB according to your situation (local database, database on Neptun, inside campus or outside).
3. Create a new package called "tests" and, inside, create a class named "BasicTests" which extends TestCase (junit.framework.TestCase)
4. Create a constructor for the BasicTests class

```
public BasicTests(String name) {

 super(name);

}
```

5. Define a private field called "db":

```
private DBService db;
```

6. Convert specifications into test steps and write them down:

The first 3 test steps could be:

- Create a new pojos.Branch
- Try to save. This will fail because every branch requires a name

- Set name and try to save. This will also fail because every branch requires a phone number
  - ...
7. Implement the written test steps in a new method called testBranch()
  8. Create a suite() method within the BasicTests class and add the testBranch() method in the test suite:

```
public static Test suite()
{

 TestSuite suite = new TestSuite();
 suite.addTest(new BasicTests("testBranch"));

 return suite;

}
```

9. Add these two methods in the class:

```
protected void setUp() throws Exception {

 super.setUp();
 db = new DBService();

 db.openConnection();

 db.clearDB();

}

protected void tearDown() throws Exception {

 super.tearDown();

 db.closeConnection();

}
```

Note: setUp() method is automatically called before each test in the test suite is run. Similarly, tearDown() is called after.

## Exercise descriptions for the partner who designs pojos.Branch class

Add CacheDB.jar into the project's build path

- In eclipse, do this by right clicking on the project > Build Path > Configure Build Path ...
  - Click Add External JARs...
  - Navigate to <Cache Root Director>/Dev/java/lib/JDKxx where xx corresponds to JDK version you are using (JDK16 corresponds to JDK1.6)
  - Select CacheDB.jar
  - Click OK
11. Create a package name "pajos" and, inside, create a class named "Branch"

12. Define all attributes according to the UML class diagram except vehicles and employees
13. Create getter and setter methods for all attributes
14. Specify that "name" and "phone" are required for Branch object creation
  - o Add CacheProperty annotation above the attributes e.g.

```
@CacheProperty (required=true)
private String name;
```

Do the same for attribute "phone".

15. Specify the pattern for "phone"
  - o Add PropertyParameter annotation above the attribute "phone"

```
@PropertyParameter (name = "PATTERN", value = "0.1\\ "+"1(1n,1\\ "
\\")1.(1n,1\\ " \\",1\\ "-\\")")
```

Note: 1) The expression is actually 0.1"+"1(1n,1" ")1.(1n,1" ",1"-") but java requires that escape symbol (\) must be used. 2) The expression means 0 to 1 time the character "+", then 1 time of either one number or one blank, then one or more of the following: one digit or one hyphen or one blank

16. Specify that "phone" must be unique. This is done by making "phone" an index
  - o Add Index annotation above the class declaration (under package / import statements)

```
@Index(
 name="BranchIndex",
 propertyNames={"phone"},
 isUnique=true
)
```

17. Specify the maximum length for "phone"
  - o Add another PropertyParameter annotation above "phone"

```
@PropertyParameters({
 @PropertyParameter (name = "PATTERN", value = "0.1\\ "+"1(1n,1\\ "
\\")1.(1n,1\\ " \\",1\\ "-\\")"),
 @PropertyParameter (name = "MAXLEN", value = "20")
})
@CacheProperty (required=true)
private Time phone;
```

Note: If there's more than one @PropertyParameter, all @PropertyParameter statements must be enclosed within @PropertyParameters({ })

18. Specify the minimum value for "openingTime"
  - o Add PropertyParameter annotation above the attribute

```
@PropertyParameter (name = "MINVAL", value = "18000")
private Time openingTime;
```

Note: In Cache, %Time datatype represents seconds since midnight.

19. Specify the maximum value for "closingTime" (similar to "openingTime")

### **Exercise steps for both partners**

20. Join both classes, `pojos.Branch` and `tests.BasicTests`, in one project called CRS

21. Run the tester class as JUnit test and see if all assertions pass. If not, adjust `testBranch()` method or `Branch` class

22. Run `BasicTests` again and repeat the process until all assertions pass

Related files are provided here: [exercise 2 files](#)

## E-3: Import `pojos.Model`, `pojos.ModelTypes`, `pojos.Manufacturer`, `pojos.PaymentRate` classes

In the second exercise, we left out attribute "vehicles" in Branch class. Now we are going to add it in this exercise.

This time, your team needs to design a Vehicle class. Each vehicle belongs to a model. To speed up your learning time, Model, ModelTypes, Manufacturer, and PaymentRate classes will be provided and you only have to import them into your project by copying the java source files to the `pojos` package of your project.

### Create `pojos.Vehicle` class

Switch programming roles in your team: The partner who designed the class (`pojos.Branch`) should now work on the tests and vice versa.

#### Exercise description for the partner who designs the test class

1. Define test steps for `pojos.Vehicle` according to the specifications below
  - A vehicle is offered in exactly one Branch. It can't exist without a branch.
  - Each vehicle's license plate number is unique.
  - Test if the number of vehicles of a branch equals the number of vehicles you have assigned to this branch.
  - The `actualKm` of a vehicle cannot be negative.
  - If a branch is deleted, all vehicles belonging to that branch have to be deleted too.
2. Implement the test steps in the `tests.BasicTests` class

#### Exercise description for the partner who designs the Vehicle class

3. Create Vehicle class in package `pojos` of your old project (of exercise 2) and define all attributes according to the UML class diagram of CRS.
4. Use the learned annotations from exercise 2 to complete the specification in the test code.
5. Since a vehicle cannot exist without a branch, define `pojos.Vehicle` as a dependent class of `pojos.Branch`. Use a `@Relationship` annotation to do define this relationship.
  - Add a `@Relationship` annotation above attribute `branch` in Vehicles class

```
@Relationship(type=RelationshipType.MANY_TO_ONE,
parentChild=true, inverseClass="Branch",
inverseProperty="vehicles")
private Branch branch;
```

This means there can be only one branch associated with each vehicle. On the other hand, there can be many vehicles in a branch. Inverse class refers to a related class on the other end. Inverse property refers to the attribute name of that property in the inverse class which holds the relationship with this class.
  - Add a `@Relationship` annotation to the Branch class above the attribute "vehicles".

```
@Relationship(type=RelationshipType.ONE_TO_MANY,
parentChild=true, inverseClass="Vehicle",
inverseProperty="branch")
private ArrayList<Vehicle> vehicles;
```

"parentChild=true" is used to specify that vehicles depend on their branch,

Branch is the parent class, Vehicle the child class. The default value for "parentChild" is false.

6. Don't forget to add a relationship between Vehicle class and Model class (but without parentChild property).

### **Exercise description for both partners**

7. Join your works together in one project.
8. Compile, run the tests, and modify testVehicle() method or Vehicle class until all assertions pass.

Related files are provided here: [exercise 3 files](#)

## E-4: Import Employee, Customer, Address, PhoneNumber classes

1. Include Person, Employee, Customer, and PhoneNumber classes to the project under "pojos" package.
2. Add relationship between Employee and Branch as specified in the UML class diagram using learned annotations.
3. In Employee and Customer classes, note a new annotation as follows
  - o @Extends annotation is used along with the normal "extends" in Java

```
@Extends(className="pojos.Person")
public class Employee extends Person
```

NOTE: @Extends annotation is actually not necessary here because Cache is able to interpret the "extends" keyword in Java. However, in some cases, @Extends may be needed to override default superclass.

4. In Address class, note another new annotation as follows
  - o @Embeddable annotation is used to specify that object of the class is a serial object which means it cannot exist as a standalone object
5. Include PersonTest class in the project under "tests" package.
6. Read the source code for PersonTest to see what it does and run PersonTest.

## Add Truck, Motorcycle, Car classes as subclasses of Vehicle

7. From the UML class diagram, there are three types of vehicles: Truck, Motorcycle, and Car. Define Truck, Motorcycle, and Car classes according to the specification in the diagram. Don't forget to make them subclasses of Vehicle.
8. Run `BasicTests` to verify that everything still works.
9. Open System Management Portal. Go to Classes under Data Management.
10. Choose the correct namespace and click on Documentation for Truck, Motorcycle, and Car classes.
11. Make sure all classes extend from Vehicle and that all attributes of Vehicle are listed.
12. Note that attributes shown in bold letters are specific attributes for the subclass while attributes in italic are common attributes in the superclass.
13. In Vehicle class, add one method called `getFinalPrice`:

```
public double getFinalPrice(double oldPrice, int numKilo)
{

return oldPrice;

}
```

14. This newly added method will be common for all subclasses of Vehicle. We are going to override this method to be specific for each subclass in a later exercise.

## E-5: Class query with -> operator

Write queries with join-conditions that find the name and street of all branches where the postal code begins with 88. Show only those branches with number of vehicles offered greater than or equal to the minimum parameter. Use "group by" and "having" to achieve this.

Define 2 queries for this task using the -> Operator

- starting at the Branch table
- starting at the Vehicle table

Create another class under "tests" package called QueryTest and add the following method to the class.

```
public void branchSummary(String startPostcode, int minVehicle)
{

//insert your query code here

}
```

Finally, invoke branchSummary method from the main method. If you don't have a branch with a postal code beginning with number 88 try a different number.

## Data Population

Look at the Person class you have imported in exercise E-4.

Person has been defined to be populatable. Note the following related to data population in the code:

- For any class to be populatable, a specific annotation is needed before class declaration
- @CacheClass(populatable = true)
- @PropertyParameter named POPSPEC is used for specifying how the populated data is generated. For example,

```
@PropertyParameter(name = "POPSPEC", value = "FirstName()")
private String firstName;
```

means that "firstName" attribute should be generated using FirstName() method of %Library.PopulateUtils which is a predefined method in Cache library.

Try adding @CacheClass(populatable=true) to Branch class without specifying POPSPEC value. This would mean that Cache has to generate the data taking into account the data type and name only.

Import *PopulateTest* class into the project under "tests" package. Run *PopulateTest* and check System Management Portal to see if data is populated as expected. Since *Person* is an abstract class only *Customer* and *Employee* objects can be created. Also note how Cache handles data population for Branch class (which has no specified POPSPEC).

## E-6: Business Logic, Methods

### Import Loan, Image, and DataPopulation class.

An object of class *Loan* will be created when a customer rents a vehicle.

1. Import *Loan* class and *Image* class to your project under "pojos" package and read the class definitions carefully.
2. If you don't have them yet import *Manufacturerer* class, *PaymentRate* class, and *ModelTypes* class.
3. Connect the new classes properly to the existing classes. In most cases relationships are appropriate. However the connections from *Vehicle* class to *Image* class and from *Model* class to *PaymentRate* class are only unidirectional. The *ModelTypes* class has no connection but is only a lookup table for data population.
4. Import *DataPopulation* and *DateUtil* under "utils" package. *DataPopulation* class is used for populating the database. *DateUtil* class provides truncateDate method that sets all time related attributes of date to zero. This prevents a problem when comparing two dates. So whenever a date object has to be created, call truncateDate method on it.
5. Import *ProcessTest* under "tests" package. Read and try to understand the code.

### Modify Vehicle, Motorcycle, Car, Truck classes / Override method

1. If you have not already done define one field in Vehicle class as follows:

```
@ID(type = IDType.SYSTEM_ASSIGNED)
private String dbID;
```

Note: This serves as a placeholder for a real database ID. It will come in handy in certain situations when you would like to know the ID of the saved object.

2. Now we're going to override getFinalPrice method in Vehicle to be specific to each subclass.
  - o For a vehicle of type *Car*, the price is calculated taken into account the *payment rate, number of days, exceeding kilometers* driven, and *discount rate*.
  - o For a vehicle of type *Motorcycle*, the price is calculated taken into account the *payment rate, number of days, and discount rate* only.
  - o For a vehicle of type *Truck*, the price is calculated taken into account the *payment rate, number of days, total kilometers* driven (not just exceeding), and *discount rate*.
3. Define additional fields in *Car* class as follows:

```
private final int KILOLIMIT = 300;
private final double EXCEEDKILO = 0.05;
```

Now, when we rebuild a schema for *Car* class, these two fields will become properties too. To prevent them from becoming properties in the database, add an annotation as follows before each property's declaration.

```
@Transient
private final int KILOLIMIT = 300;
@Transient
private final double EXCEEDKILO = 0.05;
```

4. Define an additional field in *Truck* class:

```
private final double PERKILO = 0.03;
```

Don't forget to add the *@Transient* annotation too.

5. Override *getFinalPrice* method of *Vehicle* class in *Car* and *Truck* classes (*Motorcycle* class uses the parent's implementation) to calculate new price according to the given definition in step 3.

## Add Business Logic to CRS Application

1. Develop a concept (flow chart, sequence diagram, or pseudo code) for request, retrieve, and return processes. Below is the textual description of the standard process flow for the rental process. Later only the *request* process will be implemented.

### Request (will be implemented)

- Customer comes to a branch and requests a vehicle.
- Customer provides all information: type of vehicle (van, suv, enduro, etc.), request date, return date.
- Available vehicle of the requested type during the request and return date is offered to the customer along with the estimated price he has to pay (disregard exceeding driven kilometers) .
- Customer may either accept or reject the offer.
- In case the customer rejects the offer, the process ends.
- In case the customer accepts the offer, the customer's driving license is checked whether it meets the required driving license of the vehicle (appropriate license class).
- In case the customer has no required license, the process ends. Otherwise, a loan item will be created. The status is then set to "reserved".

### Retrieve (will not be implemented)

- Customer comes to a branch to retrieve the reserved vehicle.
- A loan item is updated (initial kilometer, status set to "retrieved").
- Customer then gets the vehicle.

### Return (will not be implemented)

- Customer comes to a branch to return the vehicle.
- A final price is calculated.
- Customer pays for the rental fee.
- A loan item is updated.

Implement the request process: One student in your team should develop the test code for *processRequest* method

2. Add the *testProcessRequest()* method in the *tests.ProcessTest* class, with the following specification:
  - Check if data has been populated. If not, end the process.

- Define variables for a request, e. g. *requestor*, *requestedType*, *requestDate* but with wrong data like *requestor* = null.
  - Try to create a new loan item calling *processRequest()* with the parameter values set above.
  - Make sure that no loan item object was created.
  - Create a new order object calling *processRequest()* with correct parameter values.
  - Make sure that a new loan item object has been created with the correct values.
3. Invoke *testProcessRequest()*, *testProcessRetrieve()*, and *testProcessReturn()* in the main method of *ProcessTest* class.

The other student in your team should develop the productive code

- 4. Define the method *processRequest* according to your concept of step 1. The method *processRequest* accepts a requested vehicle information and creates a new *Loan* object only if the customer accepts the offer (should be random).
- 5. Run the tests until they succeed (Run *DataPopulation* before running the test).

## **E-7: Create Graphical User Interface (GUI) for CRS**

- Only the request process will be demonstrated for the GUI.
- Here is the textual description for the process flow which will be used to design the screen
  - User selects a customer, branch, type of vehicle, features, request date, and end date
  - User presses "Search" button
  - The system searches for all available vehicles and displays the results along with the estimated prices of all vehicles
  - User selects the vehicle he would like to rent
  - The image of the selected vehicle is displayed
  - User presses "Request" button
  - The system checks if the user has a driver's license required to drive the requested vehicle
  - If the user doesn't have the required license class, display a message in the next screen informing the user about the failure
  - If the user has the required license class, loan item is created and the information is displayed in the next screen
  - User has a choice to go back to perform the request again or exit the program