OODB's

- query language issues
  - ogl standard (ODMG)

- views (logical table)
  - sharing
  - hiding data
  - ease of use

Object-oriented $\Rightarrow$ "object id" may have no meaning in application (outside OODB).

Two Approaches:

- OOPL
  - Persistent C++

Embedded Query

```
C++

[ogl statement] $\Rightarrow$ Select Object from DB where
O.name = "Jones"
```

result is a set of objects: $O_1, O_2, \ldots, O_m$

map objects from DB to PL environment
Consider $T_1$:

\[
\begin{array}{ccc}
A & B & C \\
a_1 & b_1 & 5 \\
a_2 & b_2 & 5 \\
a_3 & b_1 & 5 \\
a_2 & b_2 & 6 \\
\end{array}
\]

Select $A, B$ from $T$ where $C=5$

\[
\begin{array}{ccc}
A & B \\
a_2 & b_1 \\
a_2 & b_2 \\
a_3 & b_1 \\
\end{array}
\]

We get back part of a relation

No way to display an "object"

Select $O$
from $DB$
where $O.name = "Jones"$

Select print($O$)
from $DB$
where $O.name = "Jones"$

Views make the DB look different for users.
$\rightarrow$ Performance: Optimize DB for critical users.
$\rightarrow$ data sharing.

Create view $\bar{T}abX$ as
Select $A, B, C$ from $T$
where $F=17$

$\bar{T}abX$ is a logical table

User query:
Select $A, B$
from $\bar{T}abX$
where $C=5$

Query that is executed:
Select $A, B$
from $T$
where $F=17, C=5$
OODB's

→ need for complex data types
→ OOP's need for persistent data types
→ oo vs db

→ object id vs oodbms
  
  degrees of persistence
  - inprocedural
  - intra-program
  - persistent
  
  approaches to id

→ Persistence objects

  - by class
    - easiest to implement
    - not flexible
  
  - by creation
    - new syntax to allow creation
    - of persistent objects
  
  - by marking
    - a function is used to mark an
      object as persistent
    - delay decision
  
  - by reachability
    - one or more objects are declared persistent
    - each object that can be reached from these
      objects are persistent
locating persistent objects in a db

- give names to objects (i.e., like a file name)
  - Hashmap in java
- expose object ids or pointers to the objects
- the set of objects & allows program to iterate
  over the set to find the object.

3. most support some variation of all three

Systems tried to COPP's

Persistent C++ Systems

(COMG Standards)

- functionality is provided via class libraries
  so no language extension is required.

Two parts to the COMG C++ extension

1. C++ Object Definition Language (C++ OD1)
2. C++ Object Manipulation Language (C++ OM1)

```
od1 C++
```

```c
class Branch : public d_object {
  public:
    d_string branch_name;
    d_string address;
    d_long;
  }
```

1. Extends d_object
2. d-object is a persistent class type
4. d-string & d-long are standard types defined by COMG
more class defns

class Account : public d::Object &
private:
    d::Long balance;
public:
    d::Long account_number;
    d::Set<d::Ref<Customer>> owners;
    d::Long find_balance();
    int update_balance(d::Long delta);
};

class Person : public d::Object &
public:
    d::String name;
    d::String address;
};

class Customer : public Person &
public:
    d::Date member_from;
    d::Long customer_id;
    d::Ref<Branch> home_branch;
    d::Set<d::Ref<Account>> accounts;
};

// d::Ref::Set is a template that combines the 2.
```
int create_account_owner (String name, String address)
{
    d_database bank_db_obj;
    d_database *bank_db = &bank_db_obj;
    bank_db -> open ("Bank-DB");
    d_transaction Trans;
    Trans . begin();

    d_ref < Account > account = new (bank_db, "Account") Account;

    d_ref < Customer > cust = new (bank_db, "Customer") Customer;
    cust -> name = name;
    cust -> address = address;
    cust -> accounts . insert . element (account);
    account -> owners . insert . element (cust);

    Trans . commit ();
    bank_db -> close ();
}
```
More d_object issues

d_Extent is a template class used to access objects in a class

example

d_Extent < Customer > all_customers (bank_db);
declares all_customers to be a class extent of Customer

example

int print_customer ()
{
    d_Database bank_db_obj;
    d_Database *bank_db = &bank_db_obj;
    bank_db -> open ("Bank_DB");
    d_Transaction Trans;
    Trans.begin ();
    d_Extent < Customer > all_customers (bank_db);
    d_ExtIterator < d_Ref < Customer > > iter = all_customers;
    create_iterator ();
    d_Ref < Customer > p;
    while ( iter . next ( p ) )
    {
        print_customer ( p );
    }
    Trans . commit ( );
}
Updating objects

-> the method \texttt{make-modified()} must be called prior to changing the values of a persistent object.

Object query language: \texttt{C++}

Ex:
\begin{verbatim}
d_set<d_Ref<Account>> result;
d_QQL_Query q1("select a
  From Customer c, c.accounts
  where c.name = 'Jones'
  and a.find_balance() > 100");

d_qql_execute(q1, result);
\end{verbatim}

-> Similar model for Java