Independence and Interoperability in Database Systems

Stephen J. Hegner
Department of Computing Science
Umeå University
Sweden

Independence vs. Interoperability

Independence: the ability to alter a basic design feature without the need to alter other design features.

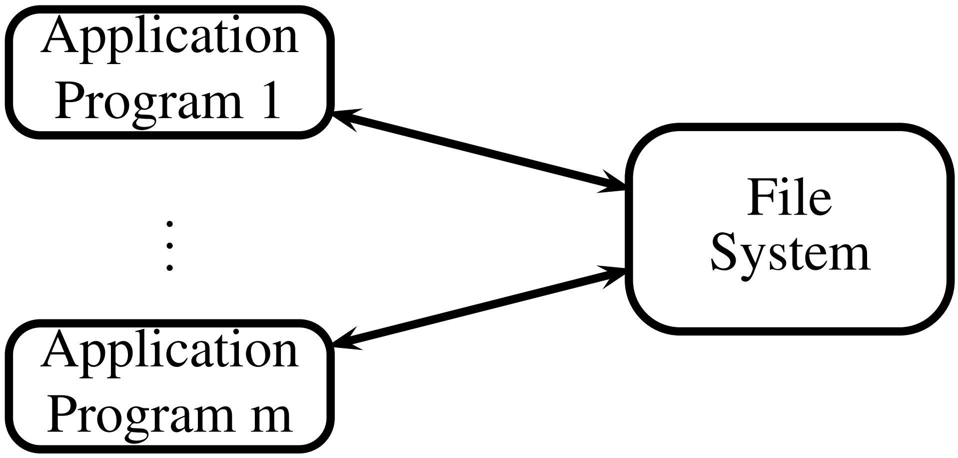
- Physical database design (underlying data structures)
- The underlying conceptual data model of a fixed database
- The host programming language of a fixed application

Interoperability: the ability to use the same applications with a variety of members of the supporting cast, including but not limited to:

- the vendor and version of the database system;
- the vendor and version of the operating system;
- the vendor and version of the program development environment.

Direct File Access

- In the classical one-level architecture, the application programs interact directly with the file system.



☟: All applications programs must be rewritten if:

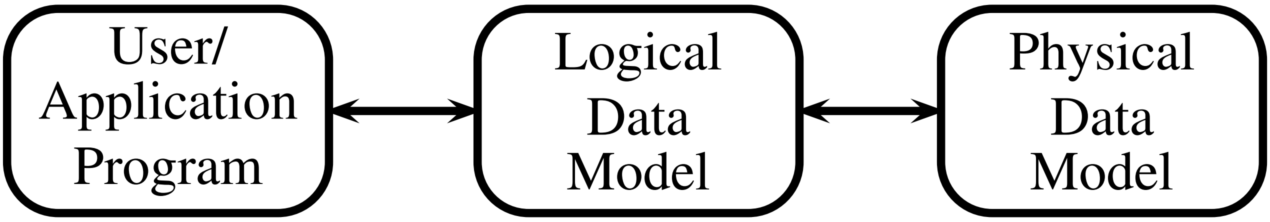
- the operating system or the hardware is to be changed; or
- the data representation is to be altered.

☟: Concurrent access is only possible to the extent that locking etc., are supported in the operating system, and then each application program must handle this function individually.

☟: This approach provides absolutely no independence.

The Two-Level DBMS Architecture

- In a two-level DBMS architecture, the application is separated from the physical data model via a logical data model.



- The logical data model may be either vendor-supplied or standardized.

Examples of vendor-supplied logical models:

classical: The IMS/VS hierarchical DBMS

modern: Most object-oriented database systems

Examples of standardized logical models:

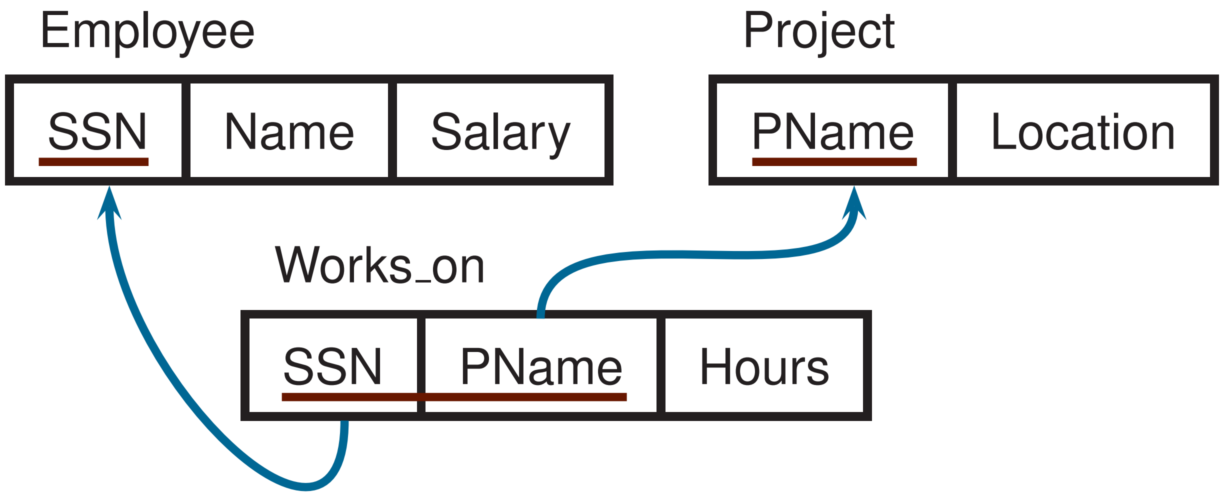
classical: The CODASYL network model

modern: The relational model

: If the physical data model is altered for any reason, only the mapping between it and the logical data model need be redesigned.

The Relational Model — an Industry Standard

- In the relational model, the data are stored in tables.
- The structure of these tables is specified via a *relational schema*.
- A toy schema:



- Key constraints (shown underlined in sepia) specify those fields which uniquely determine a tuple.
- Foreign key constraints (represented as arrows in midnight blue) specify inclusion of key fields.

A Relational Database for the Toy Schema

Employee

SSN	Name	Salary
3141592654	Kari Nordmann	80000
1618033989	Ola Nordmann	90000
2718281828	Renée Françoise	50000

Project

PName	Location
Restoration	Olso
Research	Frankfurt

Works_on

SSN	PName	Hours
3141592654	Restoration	30
3141592654	Research	30
1618033989	Research	40
2718281828	Restoration	40

Non-Procedural Queries in the Relational Model

- Function-free first-order logic with equality provides a near-perfect mathematical foundation for the relational model.

- In particular, queries may be expressed via formulas in an associated logic, called the *tuple calculus*.

Query: Find the names of those employees who work on some project which is located in Frankfurt.

$$
\begin{aligned}
&\{(e.\mathsf{Name}) \mid \mathsf{Employee}(e) \land \\
&\qquad (\exists p)(\exists w)(\mathsf{Project}(p) \land \mathsf{Works\_on}(w) \land \\
&\qquad\qquad (e.\mathsf{SSN} = w.\mathsf{SSN}) \land (p.\mathsf{PName} = w.\mathsf{PName}) \land \\
&\qquad\qquad (p.\mathsf{Location} = \text{“Frankfurt”}))\}
\end{aligned}
$$

Query: Find the names of those employees who work on every project.

$$
\begin{aligned}
&\{(e.\mathsf{Name}) \mid \mathsf{Employee}(e) \land \\
&\qquad (\forall p)(\exists w)(\mathsf{Project}(p) \ \Rightarrow \ (\mathsf{Works\_on}(w) \land \\
&\qquad\qquad ((e.\mathsf{SSN} = w.\mathsf{SSN}) \land (p.\mathsf{PName} = w.\mathsf{PName})))\}
\end{aligned}
$$

SQL — The Standard Query Language

- SQL is the standard query language which is used in virtually all relational database systems.
- It is an outgrowth of the SEQUEL project of IBM in the 1970's.
- SEQUEL = *S*tructured *E*nglish *QUE*ry *L*anguage.
- Unfortunately, SQL is not faithful to the simple and elegant query model provided by the tuple calculus.
- Rather, it is a *mélange* of several abstract query models and a great deal of *ad hoc* constructs.
- Consequently, the expression of queries is often needlessly complex and nonintuitive.
- SQL also supports:
  - ➪ Updates to the database;
  - ➪ Data definition;
  - ➪ Authorization.

Examples of SQL

Query: Find the names of those employees who work on some project which is located in Frankfurt.

```
Select  Name
From    Employee, Project, Works_on
Where   (Employee.SSN = Project.SSN)
        and (Project.PName = Works_on.PName)
        and (Project.Location = "Frankfurt");
```

Query: Find the names of those employees who work on every project.

```
Select  Name
From    Employee
Where   Not Exists
        (Select PName From Project
                Except
                (Select PName
                From Works_on
                Where (Employee.SSN = Works_on.SSN)));
```

The Rôle of SQL

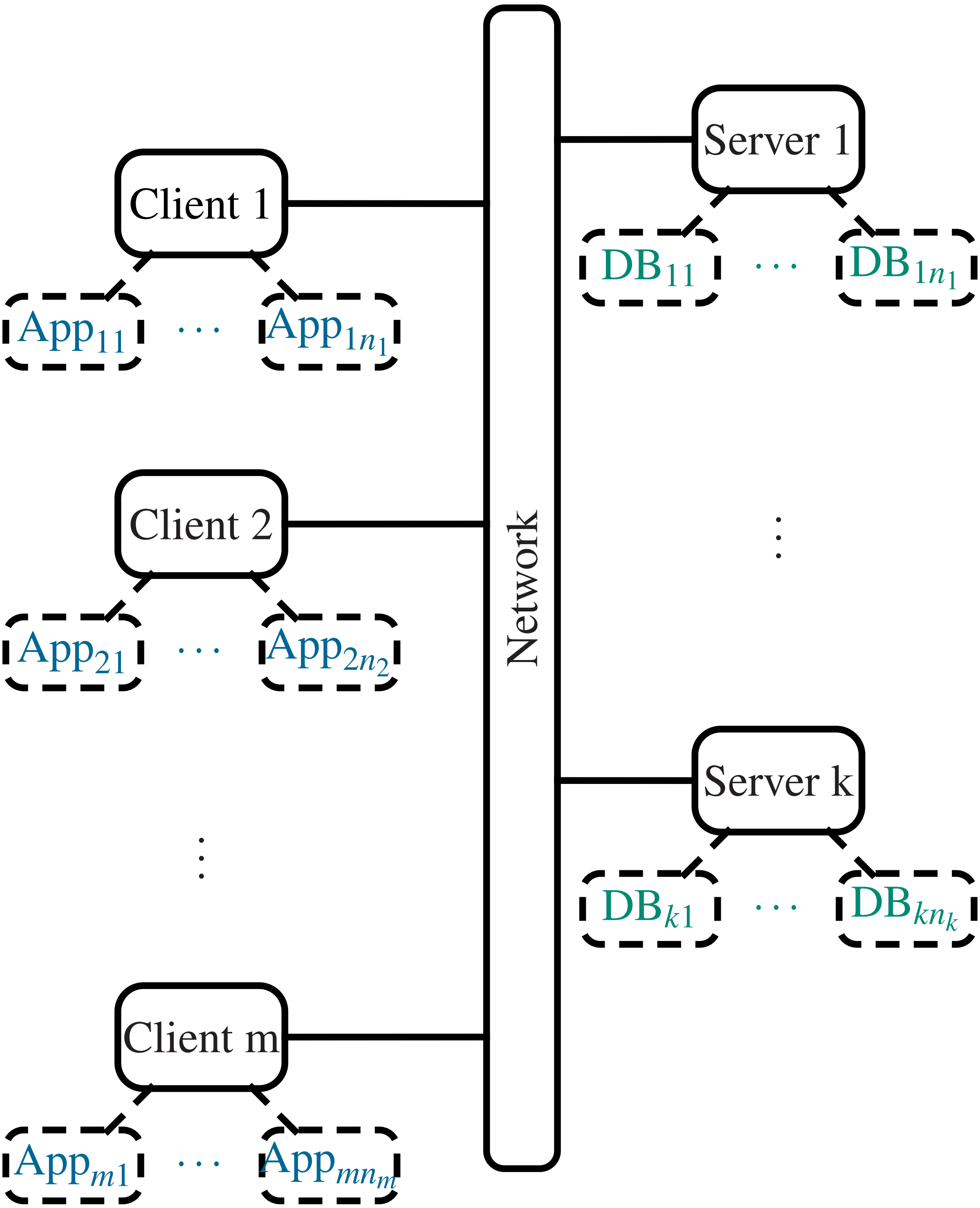
- SQL may be used as a direct user interface to a database system in simple situations.

- All systems come with a client-side program which permits the user to enter SQL queries, and receive responses, directly in a program window.

- However, it is not suitable, by itself, as a general database-application programming language, for the following reasons.
  - ➪ It is often necessary to perform complex computations on retrieved data.
    - Such computations are often impractical to express in SQL.
  - ➪ SQL is not universally suitable as a user interface.
  - ➪ It is often necessary to access several databases, and to perform computations and eventual updates based upon all of these retrievals.

- For these reasons, it is essential to be able to combine the use of SQL with that of conventional programming languages.

The Client-Server Model and Multi-DBMS's



Vendor-Specific Solutions to DB Programming

Representative example: Oracle PL/SQL

- It is a proprietary PL/1-like language which supports the execution of SQL statements which are specified in the program.
- Oracle provides the entire development environment for a variety of platforms.

Advantages:

Features: Many vendor-specific features, not common to other systems, are supported.

Performance: Performance of the executable may be optimized to the database systems of the vendor.

Disadvantages:

DBMS dependence: Any application developed with such a product is strongly bound to a specific DBMS.

Potential client platform/OS dependence: Since the development environment itself is supplied by the vendor, it may not be available for all client-side Platform/OS configurations.

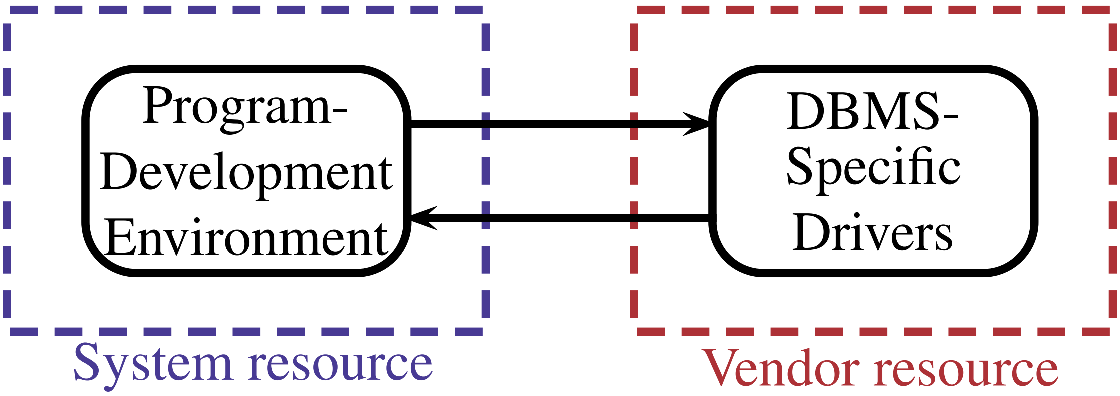
- Such solutions provide essentially no interoperability.

Cross-Vendor Solutions to DB Programming

- In cross-vendor solutions, it is typically the case that:

  - The program-development environment is generic, and not provided by the DBMS vendor.

  - DBMS drivers are provided by the DBMS vendor.



- Three alternative architectures of this configuration will be discussed:

  ➪ Embedded SQL

  ➪ Modules

  ➪ CLI/ODBC

Embedded SQL

- The program is augmented with statements of the form

```
EXEC SQL <sql-directive>
```

- A precompiler converts these to statements in the programming language which link to precompiled driver modules supplied by the DB vendor.

- The resulting program is then compiled by the extant system compiler for that language.

<u>Features</u>:

👍: The solution is independent of the DB vendor.

👍: There is an ANSI standard for embedded SQL in C.

👎: It is difficult to support more than one DB vendor in the same program.

👎: This solution depends not only upon the programming language, but upon the specific compiler. The vendor must supply a driver library for each compiler (not language) which is to be supported.

👎: It suffers from the usual problems associated with precompilers.

Support for SQL via Modules

- This approach is similar to that of embedded SQL, save that precompiler directives are replaced by:
  - function calls
  - data definitions supported by included files

☞: This approach avoids the precompiler problems associated with embedded SQL.

☞: Unfortunately, it shares most of the other problems of embedded SQL.

- Dependence upon the compiler.
- Dependence upon the DB vendor for executable modules.
- Difficulty to integrate calls to databases from distinct vendors in the same program.

☞: There is no true standard for this approach.

CLI and ODBC

- CLI = Call Level Interface
- ODBC = Open Data Base Connectivity
- These are parallel standards.
- The architecture is shown on the next slide.

<u>Features</u>:

👍: The solution is independent of the DB vendor.

👍: There is a standard for the API's.

👍: Multiple vendors are supported seamlessly.

👍: New vendors and/or databases may be added without altering anything regarding existing ones.

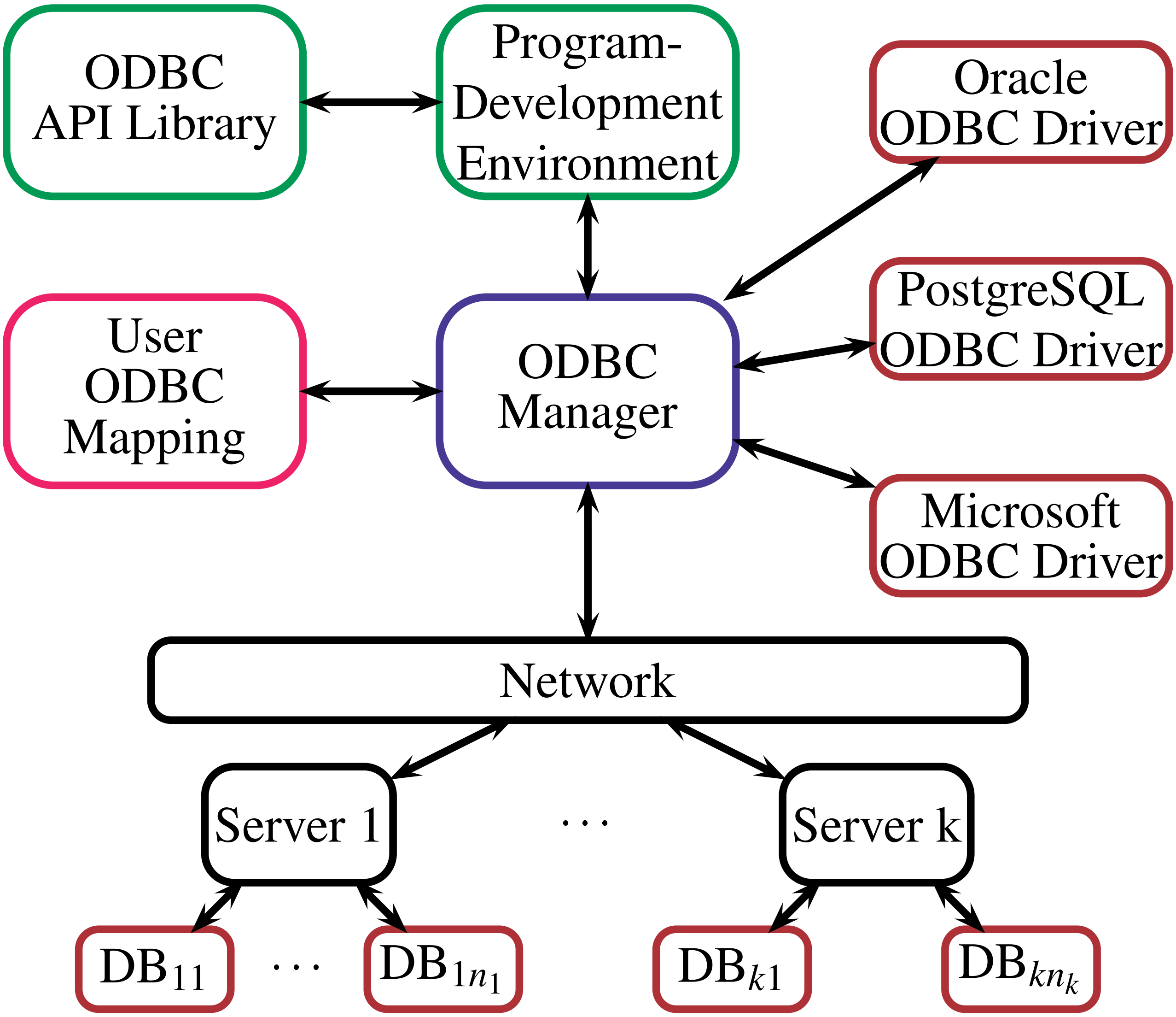
👍: The solution is independent of the programming environment.

👎: Some functionality of vendor-specific features may be sacrificed.

👎: There may be a small performance penalty over configurations which are more vendor specific.

The Architecture of ODBC

- Shown below is a typical architecture for a single client.



- Color code:
  - Supplied by the user
  - Installed in the operating system
  - Part of the development environment
  - Supplied by the system vendor

Handles in ODBC

- Just as file-access programs in a typical OS employ file handles, ODBC employs a number of types of handles.

  Environment handles: To each ODBC program is associated an environment handle, which is used to connect to the overall ODBC subsystem.

  Connection handles: To each database with which the ODBC program is to communicate is associated a connection handle.

  Statement handles: To each SQL statement which is to be compiled and shipped to a database via ODBC is associated a statement handle.

  Descriptor handles:

  - are pointers to data storage areas containing metadata which describe attributes of an SQL query, or the results of such a query;
  - are typically allocated automatically by the system for most purposes;
  - may be allocated manually for advanced operations.

The API's of ODBC

- ODBC contains over 80 API call definitions.
- Some representative calls are shown below.

API call	Description
`SQLAllocHandle`	Allocate a handle.
`SQLFreeHandle`	Release a handle
`SQLConnect`	Connect to a database
`SQLDisconnect`	Disconnect from a database
`SQLPrepare`	Compile an SQL query
`SQLExecute`	Execute a complied SQL query

Data-type mapping in ODBC

- Although ODBC is fundamentally programming-language independent, it is usually associated with C or C++.
- Shown below are some representative data mappings for these languages.
- These are defined in an include file which is usually named `sqlext.h`.
- These are used in C programs with ODBC calls to make a proper correspondence between the types of C and the corresponding ODBC structures.

Examples of ODBC $\leftrightarrow$ type C associations:

ODBC type	C type
`SQLCHAR`	`char`
`SQLINTEGER`	`long int`
`SQLREAL`	`float`
`SQLDATE`	a large `struct`

- There are also numerical encodings for the types of C and SQL, which are used only as arguments to ODBC API's.

CLI and ODBC — History and Motivation

- CLI began as an effort in parallel with SQL-92 by the SQL-Access Group, to develop a vendor-independent callable interface for SQL.

- At about the same time, Microsoft also developed a callable SQL interface, named ODBC.

- Although there are minor differences, Microsoft has always modelled its interface after CLI.

- Both have evolved greatly over the past decade.

- To switch between the two standards, usually all that is required is a change of header files.

- It seems that in the “real world,” ODBC is found more frequently than CLI, even on UNIX/Linux systems.

JDBC — A PL-Specific Alternative

- JDBC is a Java-specific alternative to ODBC.
- Like ODBC, it accommodates a variety of database vendors, who must supply JDBC drivers.
- Like ODBC, it is specific to the relational model.
- Unlike ODBC, it is tied to a single programming language.
- Unlike CLI (upon which ODBC is based), it is not an open standard, but rather a tightly controlled trademark of Sun Microsystems.

The Future — Independence and Interoperability beyond the Relational Model

Question: Why is interoperability limited to the relational model?

Answer: More modern models, such as object-oriented data models, have not matured to the point at which there is a standard query language.

Question: What about CORBA?

Answer:

- CORBA is an architecture for the brokering of objects.
- It is not specific to the database world.
- While it would be a significant player in any sort of extension of ODBC to the object-oriented world, it is not in itself such an extension.

For Further Information

- The course web page for *Databasteknik* contains:

  ➪ An annotated series of example programs written in C which perform ODBC calls.

  ➪ Slides which describe how to configure a client for proper ODBC operation.

- Consult the slides for 2002 for information on Unix/Linux clients which access the database system PostgreSQL.

- Consult the slides for 2001 for information on Microsoft Windows clients which access the database system Microsoft Access.