DATABASE TECHNOLOGY - 1DL124

Summer 2005

An introductory course on database systems

http://user.it.uu.se/~udbl/dbt-sommar05/
alt. http://www.it.uu.se/edu/course/homepage/dbdesign/st05/

Kjell Orsborn
Uppsala Database Laboratory
Department of Information Technology, Uppsala University,
Uppsala, Sweden
Personell

- Kjell Orsborn, lecturer, examiner
  - email: kjell.orsborn@it.uu.se, phone: 471 1154, room: 1321
- Tore Risch, lecturer
  - email: tore.risch@it.uu.se, phone: 471 6342, room: 1353
- Johan Petrini, course assistant
  - email: johan.petrini@it.uu.se, phone: 471 6345, room 1316
- Erik Zeitler, course assistant
  - email: erik.zeitler@it.uu.se, phone: 471 3390, room 1310
Preliminary course contents

- Course intro - overview of db technology
- DB terminology,
- ER-modeling, Extended ER
- Relational model and relational algebra
- ER/EER-to-relational mapping and Normalization
- SQL
- Transactions, Concurrency Control
- Recovery Techniques

- Storage and Index Structures
- Authorization and security
- OO/OR DBMSs
- AMOS/AMOSQL
- Query optimization
- Multimedia DBMSs
- Data warehousing
Preliminary course contents cont...

- Database assignments using Mimer SQL Engine
  - RDBMS
- Database assignment using AMOS II
  - OO/OR DBMS
- Small assignment project in AMOS II
Introduction to Database Terminology

Elmasri/Navathe chs 1-2

Kjell Orsborn

Department of Information Technology
Uppsala University, Uppsala, Sweden
The database market /CS 020524

Oracle vinnare i två klasser - dominerar Unix totalt

Oracle9i Database

DB2 Universal Database

Informix Dynamic Server (IDS)

Microsoft Access

The Office XP database solution
Evolution of Database Technology

1960
Hierarchical
(IMS)
Trees

1970
Network model
(CODASYL)
Graph

1980
Relational model
(e.g. ORACLE)
Tables

1990
Object-oriented DBMS
(e.g. ObjectStore)
OO data structures

1997
Object-relational DBMS
(e.g. SQL:99)
Object model
### STUDENT

<table>
<thead>
<tr>
<th>Name</th>
<th>StudentNumber</th>
<th>Class</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>17</td>
<td>1</td>
<td>CS</td>
</tr>
<tr>
<td>Brown</td>
<td>8</td>
<td>2</td>
<td>CS</td>
</tr>
</tbody>
</table>

### COURSE

<table>
<thead>
<tr>
<th>CourseName</th>
<th>CourseNumber</th>
<th>CreditHours</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro to Computer Science</td>
<td>CS1310</td>
<td>4</td>
<td>CS</td>
</tr>
<tr>
<td>Data Structures</td>
<td>CS3220</td>
<td>4</td>
<td>CS</td>
</tr>
<tr>
<td>Discrete Mathematics</td>
<td>MATH2410</td>
<td>3</td>
<td>MATH</td>
</tr>
<tr>
<td>Database</td>
<td>CS3380</td>
<td>3</td>
<td>CS</td>
</tr>
</tbody>
</table>

### SECTION

<table>
<thead>
<tr>
<th>SectionIdentifier</th>
<th>CourseNumber</th>
<th>Semester</th>
<th>Year</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>MATH2410</td>
<td>Fall</td>
<td>98</td>
<td>King</td>
</tr>
<tr>
<td>92</td>
<td>CS1310</td>
<td>Fall</td>
<td>98</td>
<td>Anderson</td>
</tr>
<tr>
<td>102</td>
<td>CS3220</td>
<td>Spring</td>
<td>99</td>
<td>Knuth</td>
</tr>
<tr>
<td>112</td>
<td>MATH2410</td>
<td>Fall</td>
<td>99</td>
<td>Chang</td>
</tr>
<tr>
<td>119</td>
<td>CS1310</td>
<td>Fall</td>
<td>99</td>
<td>Anderson</td>
</tr>
<tr>
<td>135</td>
<td>CS3380</td>
<td>Fall</td>
<td>99</td>
<td>Stone</td>
</tr>
</tbody>
</table>

### GRADE REPORT

<table>
<thead>
<tr>
<th>StudentNumber</th>
<th>SectionIdentifier</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>112</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>119</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>85</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>102</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>135</td>
<td>A</td>
</tr>
</tbody>
</table>

### PREREQUISITE

<table>
<thead>
<tr>
<th>CourseNumber</th>
<th>PrerequisiteNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS3380</td>
<td>CS3320</td>
</tr>
<tr>
<td>CS3380</td>
<td>MATH2410</td>
</tr>
<tr>
<td>CS3320</td>
<td>CS1310</td>
</tr>
</tbody>
</table>
Outline of a database system

DATABASE SYSTEM

Applications
procedures/statements

Users’
interactive queries

DBMS

Database language tools

Data managing tools

Database schema

Database
Database?

- A **database** (DB) is a more or less well-organized collection of related *data*.
- The information in a database . . .
  - represents information within some subarea of “the reality” (i.e. objects, characteristics and relationships between objects)
  - is logically connected through the intended meaning
  - has been organized for a specific group of users and applications
Database management system?

• A **database management system** (DBMS) is one (or several) program that provides functionality for users to develop, use, and maintain a database.

• Thus, a DBMS is a **general** software system for **defining**, **populating (creating)**, **manipulating** and **sharing** databases for different types of applications.
Database System?

• **A database system** consists of . . .
  – the physical database (instance)
  – a database management system
  – one or several database languages
    (means for communicating with the database)
  – one or several application program(s)

• **A database system** makes a *simple* and *efficient* manipulation of large data sets possible.

• The term DB can refer to both the **content** and to the **system** (the answer to this ambiguity is governed by the context).
Why DB?

• DB in comparison to conventional file management:
  - data model - data abstraction
  - meta-data - in catalog
  - program-data and program-operation independence
  - multiple views of data
  - sharing data - multiuser transactions
  - high-level language for managing data in the database
Advantages of using a database approach

- Efficient search and access of large data sets
- Controlling redundancy and inconsistency
- Access control
- Persistent storage
- Indexes and query processing
- Backup and recovery
- Multiple user interfaces
- Complex relationships
- Integrity constraints
- Active behaviour
- Enforcing standards, reducing application development time, flexibility to evolve system, up-to-date info
Data model?

• Every DB has a **data model** which makes it possible to “hide” the physical representation of data.

• A **data model** is a formalism that defines a *notation* for describing data on an abstract level together with a set of *operations* to manipulate data represented using this data model.

• Data models are used for *data abstraction* - making it possible to define and manipulate data on an abstract level.
Data models - examples

• Examples of representational (implementation) data models within the database field are:
  – Hierarchical (IMS)
  – Network (IDMS)
  – Relational (ORACLE, DB2, SQL Server, InterBase, Mimer)
  – Object-oriented (ObjectStore, Objectivity, Versant, Poet)
  – Object-relational (Informix, Oadapter, DB2)

• Conceptual data model
  – ER-model (Entity-Relationship model)
    (not an implementation model since there are no operations defined for the notation)
Meta-data, i.e. “data about data”

- Information about which information that exists and about how/where data is stored
  - names and data types of data items
  - names and sizes of files
  - storage details of each file
  - mapping information among schemas
  - constraints

- Meta-data is stored in the, so called, system catalog (or the more general term data dictionary).
Schema and instance

To be able to separate data in the database and its description the terms **database instance** and **database schema** are used.

- The schema is created when a database is defined. A database schema is not changed frequently.
- The data in the database constitute an instance. Every change of data creates a new instance of the database.
Data independence

• Reduces the connection between:
  – the actual organization of data and
  – how the users/application programs process data (or “sees” data.)

• Why?
  – Data should be able to change without requiring a corresponding alteration of the application programs.
  – Different applications/users need different “views” of the same data.
Data independence - how?

By introducing a multi-level architecture where each level represents one abstraction level.

The three-schema architecture:

In 1971 the “standard” three-schema architecture (also known as the ANSI/SPARC architecture) for databases was introduced by the CODASYL Data Base Task Group.

It consists of 3 levels:
2. Internal level
3. Conceptual level
4. External level

Each level introduces one abstraction layer and has a schema that describes how representations should be mapped to the next lower abstraction level.
Three-schema architecture

- **End users**
- **External level**
- **Conceptual level**
- **Internal level**
- **Database instance**

Conceptual schema

- View 1
- View 2
- ... (n views)

Internal schema

Database instance
Internal, conceptual and external schemas

- **Internal schema**: describes storage structures and access paths for the physical database.
  - Abstraction level: files, index files etc.
  - Is usually defined through the data definition language (DDL) of the DBMS.

- **Conceptual schema**: an abstract description of the physical database.
  - Constitute one, for all users, common basic model of the logical content of the database.
  - This abstraction level corresponds to “the real world”: object, characteristics, relationships between objects etc.
  - The schema is created in the DDL according to a specific data model.

- **External schema (or views)**: a (restricted) view over the conceptual schema
  - A typical DB has several users with varying needs, demands, access privileges etc. and external schemas describes different views of the conceptual database with respect to what the different user groups would like to/are allowed to see.
  - Some DBMS’s have a specific language for view definitions (else the DDL is used).
Views - example (Elmasri/Navathe fig 1.4)

(a) TRANSCRIPT

<table>
<thead>
<tr>
<th>StudentName</th>
<th>CourseNumber</th>
<th>Grade</th>
<th>Semester</th>
<th>Year</th>
<th>SectionId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>CS1310</td>
<td>C</td>
<td>Fall</td>
<td>99</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>MATH2410</td>
<td>B</td>
<td>Fall</td>
<td>99</td>
<td>112</td>
</tr>
<tr>
<td>Brown</td>
<td>MATH2410</td>
<td>A</td>
<td>Fall</td>
<td>98</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>CS1310</td>
<td>A</td>
<td>Fall</td>
<td>98</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>CS3320</td>
<td>B</td>
<td>Spring</td>
<td>99</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>CS3380</td>
<td>A</td>
<td>Fall</td>
<td>99</td>
<td>135</td>
</tr>
</tbody>
</table>

(b) PREREQUISITES

<table>
<thead>
<tr>
<th>CourseName</th>
<th>CourseNumber</th>
<th>Prerequisites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>CS3380</td>
<td>CS3320</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MATH2410</td>
</tr>
<tr>
<td>Data Structures</td>
<td>CS3320</td>
<td>CS1310</td>
</tr>
</tbody>
</table>
Possible data independence in the three-schema architecture

1. Logical data independence
   - The possibility to change the conceptual schema without influencing the external schemas (views).
     • e.g. add another field to a conceptual schema.

2. Physical data independence
   - The possibility to change the internal schema without influencing the conceptual schema.
     • the effects of a physical reorganization of the database, such as adding an access path, is eliminated.
Database languages

• The term *database language* is a generic term for a class of languages used for defining, communicating with or manipulating a database.

• In conventional programming languages, declarations and program sentences is implemented in one and the same language.

• A DB system uses several different languages.
  – Storage Definition Language (SDL) - internal schema
  – Data Definition Language (DDL) - conceptual schema
  – View Definition Language (VDL) - external schema
  – Data Manipulation Language (DML)
DDL and DML

• DDL is used by the database administrator and others to define *internal* and *conceptual* schema.
• In this manner the database is designed. Subsequent modifications in the design is also made in DDL.
• DML is used by DB users and application programs to *retrieve*, *add*, *remove*, or *alter* the information in the database. The term *query language* is usually used as synonym to DML.
Classification criteria for DBMSs

- Type of data model
  - hierarchical, network, relational, object-oriented, object-relational
- Centralized vs. distributed DBMSs
  - Homogeneous vs. heterogeneous DDBMSs
  - Multidatabase systems
- Single-user vs. multi-user systems
- General-purpose vs. special-purpose DBMSs
  - specific applications such as airline reservation and phone directory systems.
- Cost
Components of a DBMS (fig 2.3 Elmasri/Navathe)
Evolution of Database Technology

1960  
Hierarchical  
(IMS)  
Trees

1970  
Network model  
(CODASYL)  
Graph

1980  
Relational model  
(e.g. ORACLE)  
Tables

1990  
Object-oriented DBMS  
(e.g. ObjectStore)  
OO data structures

1997  
Object-relational DBMS  
(e.g. SQL:99)  
Object model
Classifying DBMS applications

Relational DBMS
  e.g. business data processing

Object-relational DBMS
  e.g. GIS

File systems
  e.g. text editor

Object-oriented DBMS
  e.g. ECAD

simple data  complex data

query  no query
That database market (source IDC)
Classical vs. modern DBMS applications

- Classical DBMS applications:
  - Administrative business applications, e.g. banking (ATMs)
  - Large volumes of structured data
  - Many small transactions on-line (high transaction rates)
  - High security/consistency

- Todays DBMS applications:
  - Web applications (scripting - Perl, PHP, java, web services)
  - XML data and querying
  - Multimedia data (text, graphics, images, audio, video)
  - Distributed database systems
  - Data mining
  - Data integration

- Tomorrows DBMS applications:
  - Embedded DBMSs
  - Mobile databases
  - Stream-based data and querying
  - Data grids and parallellization
  - Computational DBMSs and domain compilation
New needs for advanced database applications today and tomorrow

- **Extensibility** (on all levels - ”data blades”)
  - User types and methods (e.g. product models, geometry models, grids, matrices and spatial mathematical methods)
  - Data representations and operations (e.g. numerical matrix representations and operations)
  - Optimization techniques (e.g. specialized matrix and grid indexes, cost models and optimization algorithms)
- **Competitive performance**
  - in comparison with programming languages
- **Expressability** - complex objects
  - e.g. object-orientation needed or successor
- **Tight interfaces to programming languages**
- **New types of transactions**
  - e.g. long transactions or work in “sand box”
- **Advanced support for visualization and interaction**
  - e.g. graphical queries
Engineering database applications in Amos II

M-Sync - collaborative CAD database system

DAFES - large scale multibody systems analysis

MECHAMOS - multibody systems analysis

AMOS - product data modeling
More engineering database examples

\[ K^b \times a^f = f^b \longrightarrow \begin{cases} \ \mathbf{K}^b = (\mathbf{U}^T)^f \times \mathbf{D}^f \times \mathbf{U}^f \\ \mathbf{U}^b \times \mathbf{a}^f = \mathbf{x}^b \\ \mathbf{D}^b \times \mathbf{x}^f = \mathbf{y}^b \\ (\mathbf{U}^T)^b \times \mathbf{y}^f = \mathbf{f}^b \end{cases} \]

DECLARE K AS SymmetricMatrix;
DECLARE f AS ColumnMatrix;
SELECT a FROM ColumnMatrix a WHERE K * a = f;

FEAMOS - an embedded DBMS approach
Sample of expressing matrix algebra in AMOSQL

A sample equation from The Craig-Bampton method for component mode synthesis followed by the corresponding function that expresses the equation in AMOSQL.

\[ \hat{M}_{cy} = (K_{ff}^{-1}K_{fc})^T M_{ff} (K_{ff}^{-1}K_{fc}) - M_{cf} (K_{ff}^{-1}K_{fc}) - (K_{ff}^{-1}K_{fc})^T M_{fc} + M_{cc} \]

create function mcc(superelement se) -> submatrix as select mcc_reduced
   from   submatrix mcc_reduced, submatrix kff, submatrix kfc, submatrix mff,
   submatrix mcf, submatrix mcc
   where mcc_reduced = transpose(inverse(kff) * kfc) * mff * (inverse(kff) * kfc) -
   mcf * (inverse(kff) * kfc) -
   transpose(inverse(kff) * kfc) * mcf + mcc and
   kff = kff(se) and
   kfc = kfc(se) and
   kff = mff(se) and
   mcf = mcf(se) and
   mcc = mcc(se);
Graphical query visualization

JavaAMOS 3> display(select l from line l, body b where x(b) > 0.1 and y(b) > 0.2 and lines(b) = l);
An engineering information system - a scenario
DBMS support motivated both internally as well as externally

- **Inter-related**
  - Mediation (sharing, exchanging, constraining, transforming)
  - Distribution
  - Parallelization
  - API:s

- **Intra-related**
  - Main-memory DBMS
  - Embeddable DBMS
  - Extensibility:
    - User types and methods
    - Data representations and operations
    - Optimization techniques
  - Access to DB technology
  - OO query language and optimization
Computational applications in science

Visualization of results of an EEG simulation localizing a neural source. CACM Nov. 2004

Finite element model for studying the coupled electromechanics of the human heart. Peter Hunter, Univ of Auckland, New Zealand, CACM Nov 2004

Finite element model for studying coupled air flow, blood flow, and soft tissue mechanics in the human lung. Merryn Tawhai, Univ of Auckland, New Zealand. CACM Nov 2004

Chesapeake Bay simulation, Mary Wheeler, Univ of Texas, Austin, CACM Nov 1998
Scientific & engineering database visualization
(MIT student projects)
Projects in basic database courses
(undergraduate assignments in Uppsala)
Summing up the future of advanced database systems

- More applications moving towards 4th quadrant
  - for more complex data and query capabilities
- Database systems for mediation, distribution and parallelization
  - sharing, exchanging, constraining, and transforming data
- Data blades (cartridges/extenders) for more complex domains
  - e.g. numerical matrix algebra, symbolic formulas
- Extended querying and interface capabilities
  - such as advanced data mining, query visualization and interaction, stream querying, etc.
- New transaction models
  - e.g. long transaction and mobility
- Advanced DBMS tools for developing scientific & engineering applications
  - such as computational database applications
- Improved representations of domain models
  - product modeling, STEP, XML
Amos II three-level multi-database architecture

Clients

Mediator

Servlets

Translators

Data sources

MS Access

Java applet

FEA system

E-AMOS

ODBC

(O)JDBC

remote eval

AMOS II kernel

Optimize

Combine

Matrix module

TTR

STTR

RTR

STR

signal

ODBC

Oracle

STEP

ODBC text

AMOS II

Data sources

Clients
Mediator-Based Engineering Information Systems

Diagram:
- FEA
- E-AMOS
- CAD
- E-AMOS
- PDM
- E-AMOS
- I-AMOS
- NS-AMOS
- T-AMOS
- OODB
- S-AMOS
- RDB
- T-AMOS
- STEP file
DAFES simulation
Geographic Information Systems
Bioinformatics - NeuroGenerator
Projects in basic database courses
(undergraduate assignments in Uppsala)
Projects in basic database courses
(undergraduate assignments in Uppsala)

- Stock portfolio database with java interface
- Molecular geometry database (seen right)
Projects in basic database courses
(undergraduate assignments in Uppsala)
New DBMS Applications

- **Scientific applications**
  - Geographic Information Systems (GIS) (maps, images, spatial objects, etc.)
  - Meteorology (weather data, maps)
  - Bioinformatics (biological, chemical, genome information)
  - Astronomy (e.g. data collections from space)
  - Physics (experimental data)
- **Statistical database systems**
  - special statistical operations needed
- **Engineering applications**
  - Computer-aided design (CAD)
  - Computer-aided manufacturing (CAM)
  - Product data management (PDM)
  - Computational mechanics (Finite element analysis (FEA), Multi-body analysis (MBS))
  - Computational fluid dynamics CFD
New DBMS Applications cont. ...

- Financial analysis
  - stocks, options & derivatives
- Multi-media databases
  - text, graphics, images, audio, video
- WEB/Hypertext databases (WWW)
- Mobile databases
- Office information systems
  - structured text, graphics, audio, video
- Computer-aided software engineering (CASE)
Object-Oriented Databases

Problems with using RDBMSs for OO applications

- Complex mapping from OO conceptual model to relations
- Complex mapping => complex programs and queries
- Complex programs => maintenance problems
- Complex programs => reliability problems
- Complex queries => database query optimizer may be very slow
- Application vulnerable to schema changes
- Performance
Object-Oriented Databases

- First generation ODBs
- Extend OO programming language with DBMS primitives
  - E.g. C++, SmallTalk, Java
  - Allow persistent data structures in C++ programs
  - Navigate through database using C++ primitives (as CODASYL)
  - An object store for C++, SmallTalk, Java, etc.
- Several products out, e.g.:
  - Objectivity, Versant, ObjectStore, Gemstone, Poet, PJama, O₂
Object-Oriented Databases

• Pros and cons:
  + Long transactions with checkin/checkout model (sand box)
  + Always same language (C++)
  + High efficiency (but only for checked-out data)
  - Primitive ‘query languages’ (now OQL standard)
  - No methods in database (all code executes in client, no stored procedures)
  - Rudimentary data independence (no views)
  - Limited concurrency
  - Unsafe, database may crash
  - Slow for many small transactions (e.g. ATM applications)
  - May require extensive C++ or Java knowledge
Object-Relational Databases

- **Object-Relational DBMSs**

- **Idea:**
  - Extend on RDBMS functionality
  - Customized (abstract) data types
  - Customized index structures
  - Customized query optimizers
  - Use declarative query languages, SQL extension (SQL99)

- **Extensible DBMS**
  - Object-orientation for abstract data types
  - Data blades (data cartridges, data extenders) are database server ‘plug-ins’ that provide:
    - User definable index structures
    - Cost hints and re-write rules for the query optimizer
Object-Relational Databases

- Pros and cons:
  + Migration path to SQL
  + Views, logical data independence possible
  + Programming language independence
  + Full DBMS functionality
  + Stored procedures, triggers, constraints
  + High transaction performance by avoiding data shipping
  + Easy to use declarative queries
  - Overkill for application needing just a C++ object store
  - Performance may suffer compared to OODBs for applications needing just an object store
  - May be very difficult to extend index structures and query optimizers

- Research prototypes: Iris (HP), Postgres (Berkeley), Starburst (IBM)

- Products: Informix, OpenODB (Odapter), DB2

NOTE: On-going evolution of 1st gen. products to become more Object-Relational
\[ \dot{\mathbf{K}}_{rr} = \phi^T \mathbf{K}_g \phi \]
\[ \dot{\mathbf{K}}_{cc} = \mathbf{K}_{cc} - (\mathbf{K}_g^T \mathbf{K}_r) \mathbf{K}_c \]
\[ \dot{\mathbf{M}} = \begin{bmatrix} \dot{\mathbf{M}}_{rr} & \dot{\mathbf{M}}_{rc} \\ \dot{\mathbf{M}}_{cr} & \dot{\mathbf{M}}_{cc} \end{bmatrix} \]
\[ \dot{\mathbf{M}}_{rr} = \phi^T \mathbf{M}_g \phi \]
\[ \dot{\mathbf{M}}_{cc} = -\phi^T \mathbf{M}_g \mathbf{K}_r^{-1} \mathbf{K}_r + \phi^T \mathbf{M}_c \]
\[ \dot{\mathbf{M}}_{cr} = (\mathbf{K}_g^T \mathbf{K}_r) \mathbf{M}_g (\mathbf{K}_r^T \mathbf{K}_r) - \mathbf{M}_g (\mathbf{K}_r^T \mathbf{K}_r) - (\mathbf{K}_g^T \mathbf{K}_r)^T \mathbf{M}_c + \mathbf{M}_{cc} \]

create function myy_reduced(superelement se) -> submatrix as
select myy_reduced
from submatrix myy_reduced, submatrix mff, matrix eg
where myy_reduced = transpose(eg) * mff * eg and
mff = mff(se) and
eg = eigenvector(se);
create function myc_reduced(superelement se) -> submatrix as
select myc_reduced
from submatrix myc_reduced, matrix eg, submatrix mff,
submatrix kff, submatrix kfc, submatrix mfc
where myc_reduced = -transpose(eg) * mff *
inverse(kff) * kfc + transpose(eg) * mfc and
eg = eigenvector(se) and
mff = mff(se) and
kff = kff(se) and
kfc = kfc(se) and
mfc = mfc(se);
create function kyy_reduced(superelement se) -> submatrix as
select kyy_reduced
from submatrix kyy_reduced, submatrix kff, matrix eg
where kyy_reduced = transpose(eg) * kff * eg and
kff = kff(se) and
eg = eigenvector(se);
create function kcc_reduced(superelement se) -> submatrix as
select kcc_reduced
from submatrix kcc_reduced, submatrix kfc
submatrix kff, submatrix kfc
where kcc_reduced = kcc -
transpose(inverse(kff)*kfc)* kfc and
kcc = kcc(se) and
kff = kff(se) and
kfc = kfc(se);