Information Systems

Database System Architecture. Relational Databases

Temur Kutsia

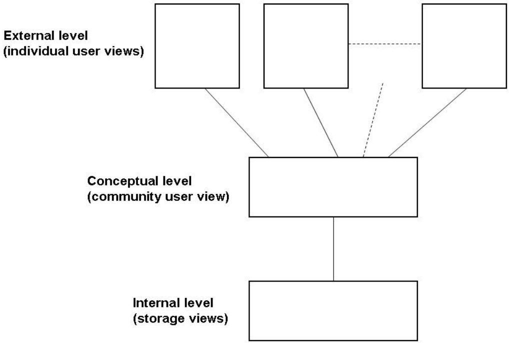
Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Outline

The Three Levels of the Architecture

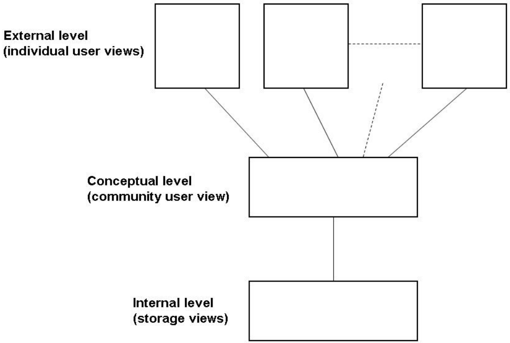
- Goal: To present an architecture of a database system.
- This will give a framework on which the subsequent material will be built.
- This architecture fits well to most of the systems.
- Three levels: Internal, conceptual, and external.

The Three Levels of the Architecture



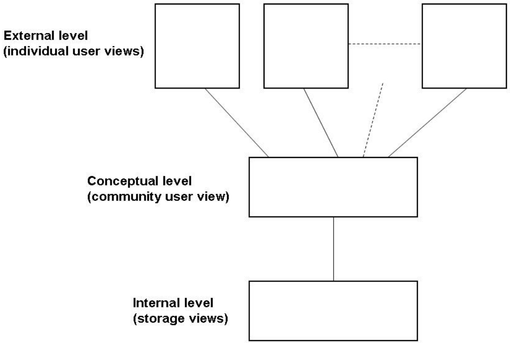
- The internal level: closest to physical storage, concerned with the way data is stored inside the system.

The Three Levels of the Architecture



- The external level: closest to users, concerned with the way the data is seen by individual users.

The Three Levels of the Architecture



- The conceptual level: a level of indirection between the other two.

The Three Levels of Architecture

External (PL/I)

```
DCL 1 EMPP,
      2 EMP# CHAR(6)
      2 SAL FIXED BIN(31)
```

External (COBOL)

```
01 EMPC.
   02 EMPNO PIC X(6).
   02 DEPTH PIC X(4).
```

Conceptual

```
EMPLOYEE
  EMPLOYEE_NUMBER    CHARACTER(6)
  DEPARTMENT_NUMBER  CHARACTER(4)
  SALARY             DECIMAL(5)
```

Internal

```
STORED_EMP  BYTES=20
  PREFIX    BYTES=6,OFFSET=0
  EMP#      BYTES=6,OFFSET=6,INDEX=EMPX
  DEPT#     BYTES=4,OFFSET=12
  PAY       BYTES=4,ALIGN=FULLWORD,OFFSET=16
```

Mappings

- Corresponding data items can have different names at different points in the scheme.
- Example: The employee number on the previous slide.
- The system must be aware of such correspondences, called mappings.

Outline

The External Level

- The external level is an individual user level.
- Each user has a language at her disposal:
  - For the application programmer, the language is either a conventional programming language (Java, C++, etc.) or a proprietary language specific to the system.
  - For an end user, the language is either a query language (probably SQL) or some special-purpose language, perhaps menu- or forms-driven.
- All these languages include data sublanguage (DSL) concerned with database objects and operations.
- One particular DSL supported by almost all current systems is SQL (to be used both a stand-alone query language and embedded in other languages).

The External Level

- Any DSL is a combination of two subordination languages: a data definition language (DDL) and a data manipulation language (DML).
- DDL supports the definition or “declaration” of database objects.
- DML supports the processing or “manipulation” of database objects.

The External Level

- The external view consists of many occurrences of many types of external records.
- The users DSL is thus defined in terms of external records.
- For instance, DML *retrieve* operation will retrieve external record occurrences, not the stored ones.
- Each external view is defined by an external schema, consisting of definitions of each of the external record types in that external view.

Outline

The Conceptual Level

- The conceptual level is a representation of the entire information content of the database.
- The form of the representation is abstract in comparison with the way in which the data is physically stored.
- The form is also, in general, different from the way the data is viewed by any particular user.
- The conceptual view is intended to be a view of the data "as it really is" rather than as users (are forced to) see it.

The Conceptual Level

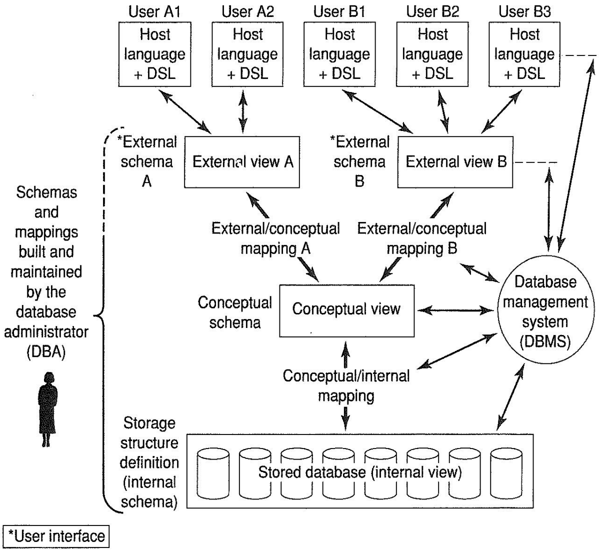
- The conceptual view consists of many occurrences of many types of conceptual records.
- Example: It might consist of a collection of department record occurrences, plus a collection of employee record occurrences, plus a collection of supplier record occurrences, and so on.
- The conceptual view is defined by means of conceptual schema, which includes definitions of each of the various conceptual record types.
- The conceptual schema is written using the conceptual DDL.
- In most existing systems the conceptual schema is little more than simple union of all the individual external schemas, plus certain security and integrity constraints.

Outline

The Internal Level

- The internal level is a low-level representation of the entire database.
- It consists of many occurrences of many types of internal records (we call them stored records).
- The internal view does not deal in terms of physical records of any device-specific considerations.
- The internal view is described by means of the internal schema.
- The internal schema defines the various stored record types, plus specifies what indexes exist, how stored fields are represented, what physical sequence of stored records are in, and so on.
- The internal schema is written using the internal DDL.
- Other terms for internal view and internal schema: stored database and stored database definition, respectively.

Detailed Architecture



Summary

- Database system architecture consists of three levels.
- The internal level is the one closest to physical storage.
- The external level is the one closest to the users.
- The conceptual level is a level of indirection between these two.
- The data as perceived at these levels is defined by a schema or schemas.
- Mappings define correspondence between
  - a given external schema and the conceptual schema, and
  - the conceptual schema and internal schema.
- Users interact with the data by means of DSL.
- DSL consists of at least two subcomponents: DDL and DML.

Outline

An Informal Look at the Relational Model

- Relational model provides the theoretical foundations of relational systems.
- Intuitive and informal introduction to relational databases.

An Informal Look at the Relational Model

Relational model has the following three aspects:

- Structural aspect: The data is perceived as tables.
- Integrity aspect: The tables satisfy certain integrity constraints (considered a bit later).
- Manipulative aspect: Operators that manipulate tables derive tables from tables.

Example: Restrict, Project, Join Operations

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Example: Restrict, Project, Join Operations

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Restrict: DEPTs where BUDGET > 8M

Result:

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M

Extracts specified rows from the table.

Example: Restrict, Project, Join Operations

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Project: DEPTs over DEPT#, BUDGET

Result

DEPT#	BUDGET
D1	10M
D2	12M
D3	5M

Extracts specified columns from the table.

Example: Restrict, Project, Join Operations

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Join: DEPTs and EMPs over DEPT#

Result

DEPT#	DNAME	BGT.	EMP#	ENAME	SAL.
D1	Marketing	10M	E1	Lopez	40K
D1	Marketing	10M	E2	Cheng	42K
D2	Development	12M	E3	Finzi	30K
D2	Development	12M	E4	Saito	35K

Combines the tables based on common values in a common column.

Structural and Manipulative Aspects

- Operations operate on tables and derive tables: Closure property of relational systems.
- Closure property is very important: The output of one operation can become input to another.
- Nesting relational expressions: Projection of a join, join of two restrictions, etc.

Structural and Manipulative Aspects

Two additional points:

1. Relational systems require the database to be perceived by the user as tables: Logical (not physical) structure.
2. Relational systems abide The Information Principle: The entire information content of the database is represented in one and only one way—as explicit values in column positions in rows in tables.

Integrity constraints

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Integrity constraints

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Examples of integrity constraints:

- Employee salaries might have to be in the range 25K to 95K.
- Department budgets might have to be in the range 1M to 15M.

Integrity constraints

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

Some integrity constraints are very important and enjoy some special nomenclature. Example:

- Each row in the table DEPT must include a unique DEPT# value.
- Each row in the table EMP must include a unique EMP# value.
- The DEPT# column in DEPT is a primary key for the DEPT table.
- The EMP# column in EMP is a primary key for the EMP table.

Integrity constraints

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finzi	D2	30K
E4	Saito	D2	35K

More constraints of the similar fashion:

- Each DEPT# value in EMP must exist as a DEPT# value in DEPT: Every employee must be assigned to an existing department.
- The DEPT# column in EMP is a foreign key, referencing the primary key of table DEPT.

Outline

Types

- Type is a set of values.
- Examples: INTEGER (the set of all integers), CHAR (the set of all character strings), S# (the set of all supplier numbers), WEEKDAY (Monday–Sunday).
- Types are also called domains.
- Types are either system-defined (built-in) or user-defined.
- Any type can be used as the basis for declaring relational attributes.
- Purpose of types: To constrain values

Types

- Any given type has an associated set of operators.
- For the system-defined type INTEGER:
  - The system provides operators “=”, “<”, and so on, for comparing integers.
  - It also provides operators “+”, “*”, for performing arithmetic on integers.
  - It does not provide operators like “||” (concatenate), SUBSTR (substring), so on, for performing string operations on integers.
- For the user-defined type S#:
  - We would probably define operators “=”, “<”, and so on, for comparing supplier numbers.
  - We would probably not define operators “+”, “*”, for performing arithmetic on supplier numbers.

Values vs Variables

- Value: an individual constant (e.g. the integer 3).
- A value can not be updated.
- Variable is a holder for an appearance of a value.
- Variables can be updated: the current value of the variable in question can be replaced by another value.

Values, Variables, Types

- Every value is of some unique type which never changes.
- Distinct types are disjoint.
- Every variable is explicitly declared to be of some type.
- Every attribute, operator, parameter of an operator is explicitly declared to be of some type.
- Every expression is at least implicitly declared to be of some type (of the type declared for the outermost operator).

Scalar vs Nonscalar Types

Any given type is either scalar or nonscalar.

- A nonscalar type is a type whose values are explicitly defined to have a set of directly accessible components.
- Otherwise, the type is scalar.
- Values, variables, attributes, operators, parameters, expressions are scalar or nonscalar depending on the corresponding type.

Scalar vs Nonscalar Types

- Values of type *T* must have at least one possible interpretation (declared as part of the definition of type *T*).
- Distinguish between accessible components of a type and accessible components of its representation.
- Type may be scalar, but its possible representation may have accessible components.
- Type definition (in a relational language):

  TYPE QTY POSSREP { INTEGER } ;
- QTY is scalar, but its possible representation has an accessible component, of type INTEGER.

Scalar vs Nonscalar Types

- Another example of type definition:
  TYPE Point
  POSSREP CARTESIAN { X RATIONAL, Y RATIONAL }
  POSSREP POLAR { R RATIONAL, θ RATIONAL };
- Two distinct possible representations.
- Each of these representations has two accessible components, of type RATIONAL.
- POINT is of scalar type: does not have accessible components.
- Nonscalar type definitions come later.
- Types can be defined in terms of user-defined typed:
  TYPE LINESEG POSSREP { BEGIN POINT, END POINT }

Type Operators

Each POSSREP declaration causes automatic definition of two operators:

- A selector operator: Allows the user to specify or select a value of the type by supplying a value for each component of the possible representation.
- A set of THE_ operators (one for each component of possible representation): Allows to access the corresponding possible-representation components of values of the type.
- Selectors have the same name as the corresponding possible representation.
- THE_ operators have the name THE_C, where C is the name of the corresponding component of the corresponding possible representation.

Type Operators

Example

- CARTESIAN (5.0, 2.5)
- CARTESIAN (X1, Y1)
- POLAR (2.7, 1.0)
- THE_X (P) (The X coordinate of the point in P, where P is a variable of type POINT)
- THE_Y (exp) (The Y coordinate of the point denoted by the expression exp).

Type Definitions

Two ways of type definitions:

- By a TYPE statement.
- By a type generator.

TYPE statement

Example:

- TYPE WEIGHT POSSREP { D DECIMAL (5,1)
  CONSTRAINT D > 0.0 AND D < 5000.0 };
- Weights can possible be represented by decimal number of five digits precision with one digit after the decimal number.
- The decimal number is greater than 0 and less than 5000.

Operators

- Two kinds: read-only and updates.
- Definition of read-only operators involves RETURNS specification.
- Definition of update operators involves UPDATES specification.

Read-Only Operators

Examples:

- ```
  OPERATOR ABS ( Z RATIONAL ) RETURNS RATIONAL ;
     RETURN ( CASE
        WHEN Z ≥ 0.0 THEN +Z
        WHEN Z < 0.0 THEN -Z
     END CASE ) ;
  END OPERATOR ;
  ```
- ```
  OPERATOR GT ( Q1 QTY, Q2 QTY ) RETURNS BOOLEAN
     RETURN ( THE QTY ( Q1 ) > THE_QTY ( Q2 ) ) ;
  END OPERATOR ;
  ```

Update Operators

Example:

- ```
  OPERATOR REFLECT ( P POINT ) UPDATES P ;
     BEGIN ;
        THE_X ( P ) := – THE_X ( P )
        THE_Y ( P ) := – THE_Y ( P )
        RETURN
     END ;
  END OPERATOR ;
  ```
  ```

Type Generators

- Type generator is an operator that returns a type.
- ARRAY INTEGER [8]
  ARRAY type generator generates an array of integers of size 12.
- VAR SALES ARRAY INTEGER [8]
  Declares variable SALES of type ARRAY INTEGER [8].
- Selector and THE_ operators exist for generated types:
  ARRAY INTEGER (2, 5, 3, 67, 23, 12, 32, 78)
  SALES [3]
- Assignment and equality comparison operators also apply:
  SALES := ARRAY INTEGER (2, 5, 3, 67, 23, 12, 32, 78)
  SALES = ARRAY INTEGER (2, 5, 3, 67, 23, 12, 32, 78)

Relations

- Up to now we discussed type, values, and variables in general.
- Next: Relations types, values, and variables in particular.
- Will continue on the next lecture.