Part 5: Logical Design II

References:

- Teorey: Database Modeling & Design, 3rd Edition.
  Morgan Kaufmann, 1999, ISBN 1-55860-500-2, ca. $32.
- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Ed.
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German), Teubner, 1997.
- Kemper/Eickler: Datenbanksysteme (in German), Oldenbourg, 1997.
- Graeme C. Simsion, Graham C. Witt: Data Modeling Essentials, 2nd Edition.
  Coriolis, 2001, ISBN 1-57610-872-4, 459 pages.
- Barker: CASE*Method, Entity Relationship Modelling.
  Addison-Wesley, 1990, ISBN 0-201-41696-4, ca. $61.
- Koletzke/Dorsey: Oracle Designer Handbook, 2nd Edition.
  ORACLE Press, 1998, ISBN 0-07-882417-6, ca. $40.
- A. Lulushi: Inside Oracle Designer/2000.
  Prentice Hall, 1998, ISBN 0-13-849753-2, ca. $50.
- Oracle/Martin Wykes: Designer/2000, Release 2.1.1, Tutorial.
  Part No. Z23274-02, Oracle, 1998.
- Oracle Designer Model, Release 2.1.2 (Element Type List).
- Oracle Designer Online Help System.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

- explain the steps in which a database schema is developed with Oracle Designer and name the tools that are used in this process.
- write a short paragraph about the Database Design Transformer of Oracle Designer: What it can do and what its limitations are.
- read Server Model Diagrams in Oracle Designer.

Overview

1. Database Design Transformer
2. Design Editor: Server Model Diagrams
3. Design Editor: Database Administration
4. Generation of SQL Code

Development Steps (1)

- First (during the conceptual design phase), one develops ER-diagrams with the ER-Diagrammer.

  The Repository Object Navigator can be used to check the global schema (and alter it, if necessary).

  Actually, one might start with business process diagrams and then design application program functions and the ER-schema concurrently.

- Then the Database Design Transformer is used to translate the ER-Schema (as stored in the Repository) into the relational model.

  One can choose to either translate the global schema or subsets of it step by step. The first option seems clearer.

Development Steps (2)

- The resulting relational schema is stored in the repository.
- One can then edit the relational schema (with the Design Editor or the Repository Object Navigator).
  - ◇ E.g. rename certain tables and columns.
  - ◇ View definitions, indexes, triggers, and other information that is not present in the ER-schema can be added at this stage.

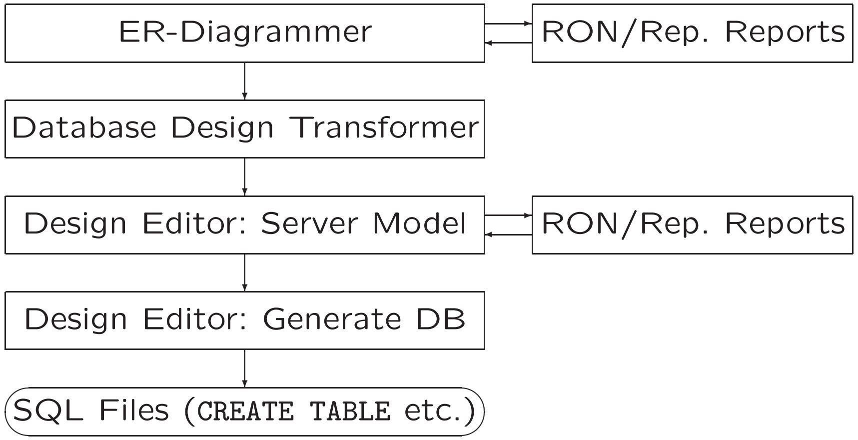
    The DB Design Transformer does not generate certain constraints that would be necessary for an exact translation of the given ER-schema. These must be added manually in this step.

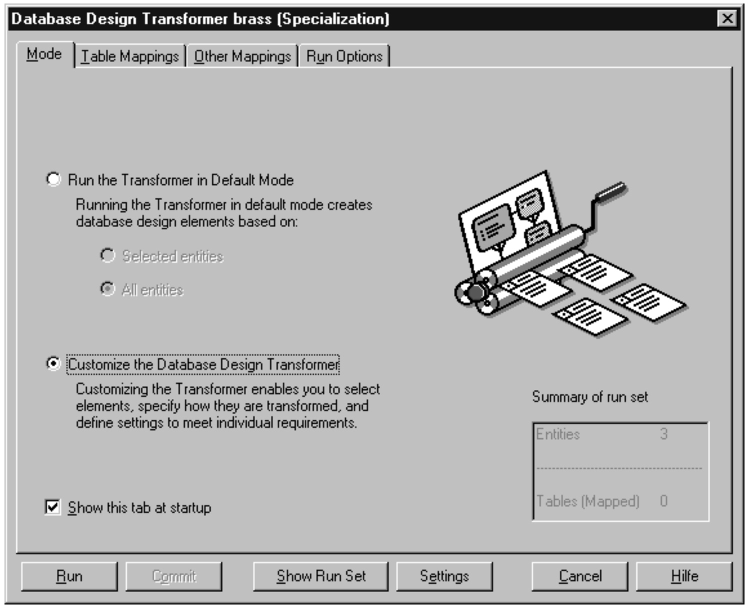
Development Steps (3)

- In the Design Editor, "Server Model Diagrams" can be developed that are a graphical representation of the relational schema.
- Finally, one can generate SQL code (for various database management systems) from the definitions stored in the repository.

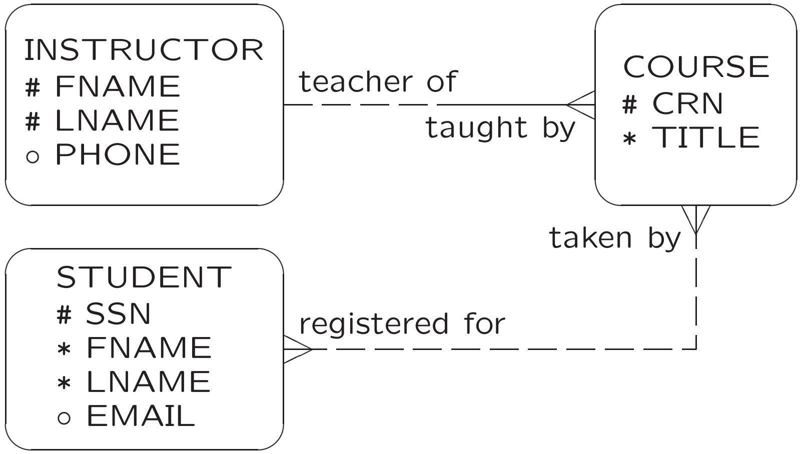
  This is also done with the Design Editor.

Development Steps (4)





Example



DB Design Transformer (1)

- As explained above, each entity type is transformed into a table:
  - ◇ The plural form of the entity type name is used as table name.
  - ◇ Spaces and punction characters in entity type and attribute names are mapped to underscores.
  - ◇ If a name is a reserved word in SQL, that name is modified (e.g. FROM becomes FROM_FROM).

    Reserved words depend in part on the DBMS, which is a problem in this step.

DB Design Transformer (2)

- Attributes of the entity type are translated into columns of the corresponding table:

  INSTRUCTORS(<u>FNAME</u>, <u>LNAME</u>, PHONE°).
  STUDENTS(<u>SSN</u>, FNAME, LNAME, EMAIL°).

- Columns are optional (null values allowed) if the corresponding source attribute is optional.

  The ER-Diagrammer permits optional attributes in primary keys. The DB Design Transformer silently corrects this mistake and makes the column not optional. Alternate key attributes remain optional.

- Primary/Alternate keys (UIDs) of the entity type become primary/alternate keys of the table.

DB Design Transformer (3)

- If an entity type has no primary key, a surrogate key is automatically added.
- E.g. for the INSTRUCTOR entity type, an attribute INST_ID of type NUMBER(10) would be added (where INST is the short name).

  In addition, a sequence called INST_ID is generated (for producing unique numbers). One can choose the domain for the ID columns.

- An option of the Database Design Transformer is to create a surrogate key for each table in this way.

  Then the declared primary keys for the entity types become alternate keys for the tables.

DB Design Transformer (4)

- For one-to-many relationships, foreign keys are added to the table at the "many" side (as expected):

```
COURSES(CRN, TITLE,
        (INST_FNAME, INST_LNAME) → INSTRUCTORS)
```

  The Database Design Transformer is able to produce foreign keys consisting of more than one column as in this case.

- Foreign key column names are constructed from the short name of the referenced entity type and the name of its primary key attribute.

  One can choose whether one wants the prefix. If the surrogate primary keys already have prefixes, one gets names like `INST_INST_ID`.

DB Design Transformer (5)

- If there name clashes (the table already has a column of that name), column names are made unique by adding the name of the relationship end (if that still does not help, numbers are added).

- If the participation in the relationship is optional, the foreign key attributes are declared as optional.

  However, if the foreign key consists of more than one attribute, a check constraint should be added that they can only be both null, or both not null. But the Database Design Transformer does not generate such constraints.

DB Design Transformer (6)

- If relationships are mutually exclusive (participate in an arc), the corresponding arc is stored for the generated foreign keys.

  However, when SQL code is later generated, the corresponding `CHECK` constraint is missing.

- One can choose how the generated foreign keys behave in case of deletions of the referenced row (restrict, cascade, nullify).

  One can also choose what happens in case of updates of the primary key values of the referenced row. The default value is "restrict" for updates and deletes, i.e. the deletion or update is not possible.

DB Design Transformer (7)

- For many-to-many relationships, an intersection table is generated:

```
COURSES_STUDENTS(CRS_CRN→COURSES,
                 STUD_SSN→STUDENTS).
```

- The name of the table for the relationship is composed out of the plural forms of both entity types.

  Probably it would have been nicer if the relationhip names were used in some way. It is possible to change generated table and column names later in the Design Editor. Also, I would have preferred "`STUDENTS_COURSES`", with the "from" side of the relationship first. But the Database Design Transformer always uses the alphabetic sequence.

Restrictions (1)

- The alternate keys that would enforce one-to-one relationships are not generated.

  One-to-one relationships are translated by the DB Design Transformer in the same way as one-to-many relationships.

- Mandatory participation for many-to-many relationships or on the "one" side of one-to-many relationships are also lost in the translation.

  As explained above, this is no fault of the Database Design Transformer, since there is no good translation.

- No warning is generated.

Restrictions (2)

- Of the nine types of relationships that can be used in the ER-diagrammer, only three are exactly translated (see next page), the other ones are approximated by more liberal relationship types.

  As explained above, it would have been possible to implement also the three kinds of one-to-one relationships (except recursive ones).

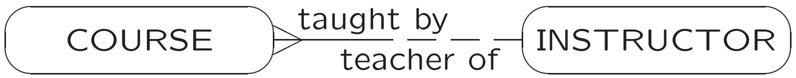
- Even constraints that cannot be enforced declaratively in the `CREATE TABLE` statements should be documented in the repository.

  The DB Design Transformer does not generate such constraints for the problematic cardinalities.

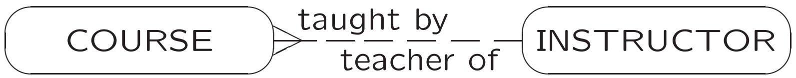
Restrictions (3)

Exactly Translated Relationship Types:

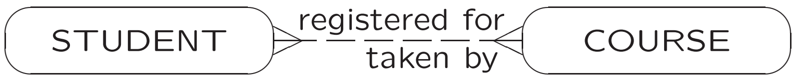
- Many-to-one, mandatory-to-optional:

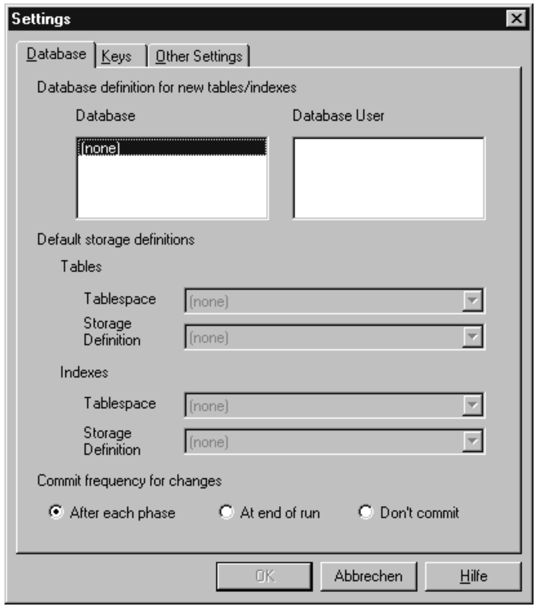


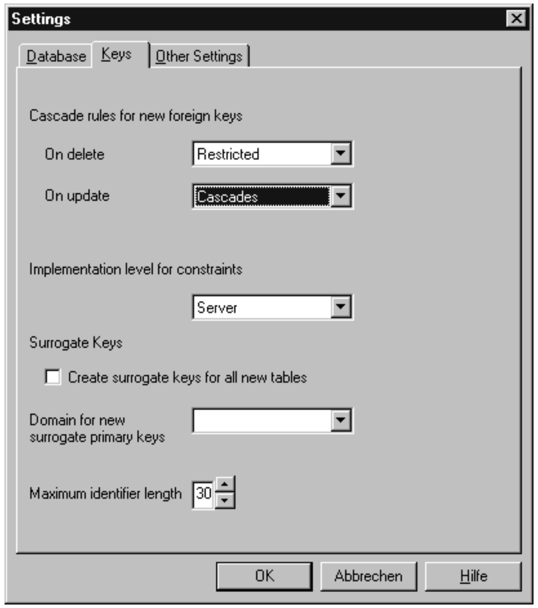
- Many-to-one, optional-to-optional:

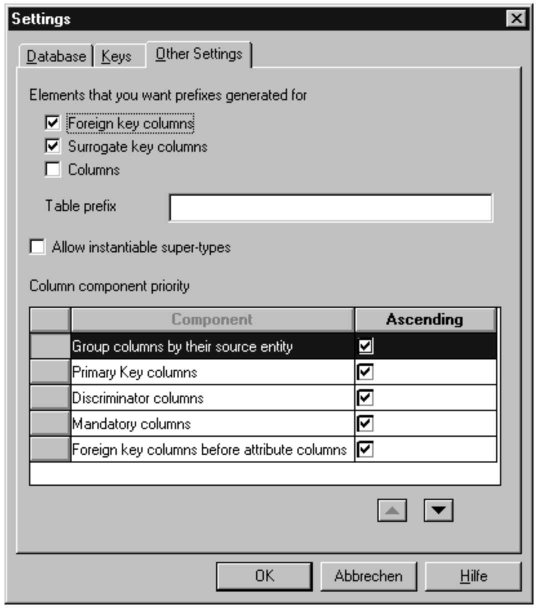


- Many-to-many, optional-to-optional:



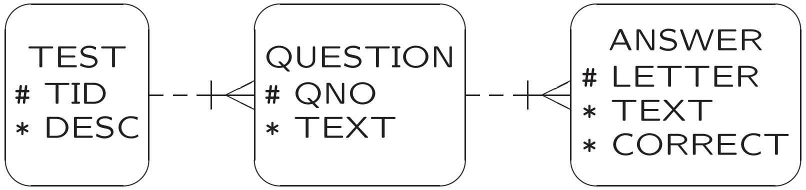






Translation of Weak Entities

- The DB Design Transformer translates a hierarchy of weak entities as expected:




```
TESTS(TID, TEST_DESC) -- DESC is a reserved word
QUESTIONS(TEST_TID→TESTS, QNO, TEXT)
ANSWERS((QUEST_TEST_TID, QUEST_QNO)→QUESTIONS,
        LETTER, TEXT, CORRECT)
```

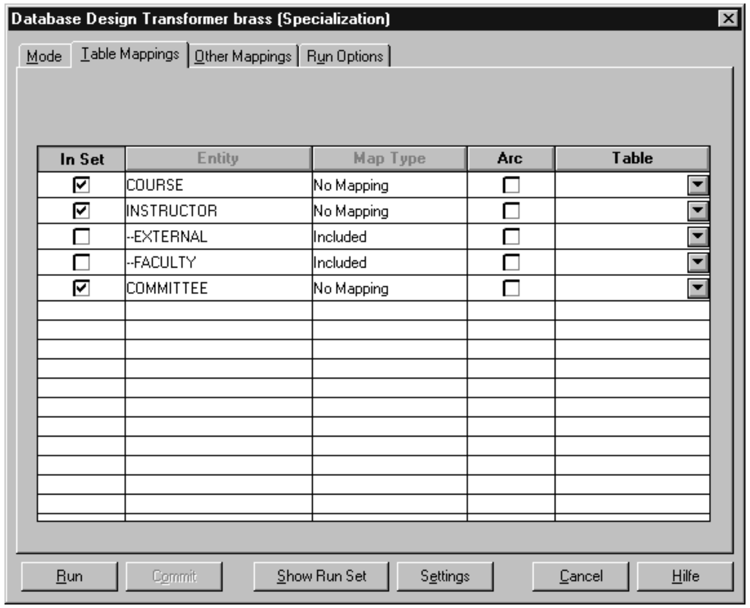

Translation of Subtypes (1)

- The Database Design Transformer supports
  - ◇ Method 1 ("Single Table Approach")

    This is the default. One can specify in the Database Design Transformer which entity types are mapped to tables. Method 1 means that only the supertype is mapped to a table, the subtypes are marked as "Included".

  - ◇ Method 2 ("Separate Table Approach")

    One gets this transformation if one selects the subtypes to be mapped to tables, but not the supertype. Select the radio button "Customize the Database Design Transformer". Then the tab "Table Mappings" appears. There select the "In Set" checkbox for the subtypes and deselect it for the supertype.

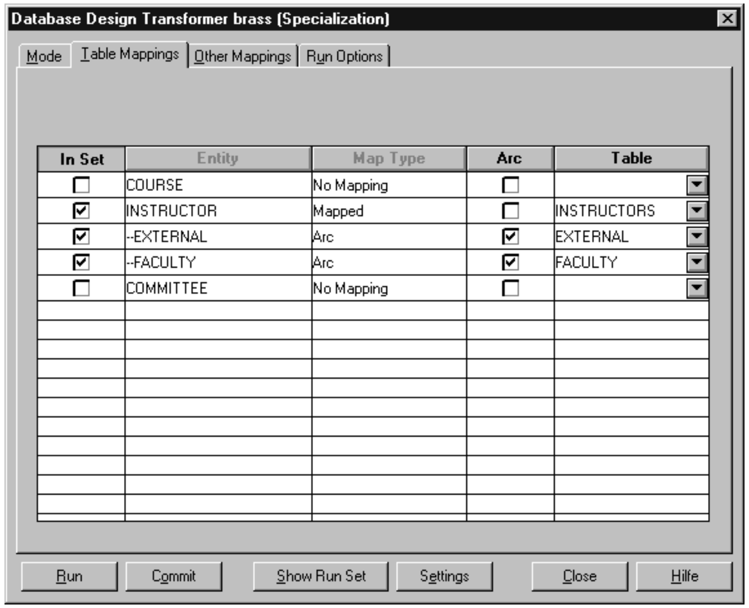


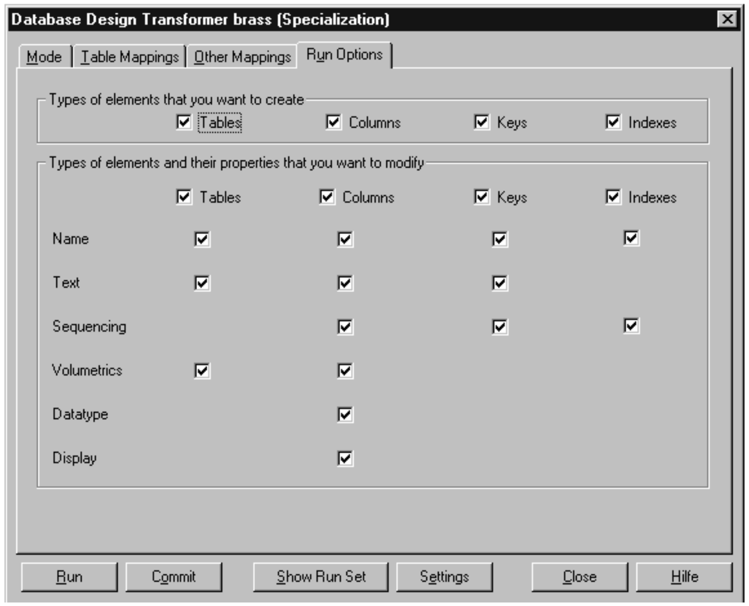
Translation of Subtypes (3)

- Supported Translation Methods, continued:
  - ◇ Method 2, Variant for Partial Specialization (“Implicit Sub-Type Approach”)

    One gets this option if supertype and subtype are mapped to tables. Instantiable supertypes can be selected in “Settings/Other Settings”. But one gets this translation also if it is not selected.
  - ◇ Method 4 (“Arc Approach”).

    For this transformation, the DB Design Transformer must be started two times (only for the supertype and the subtype, other entity types should be mapped in a third run): First map supertype and subtypes (check “In Set”) but in the “Run Options” permit only the generation of tables, not of columns or keys. In the second run, permit to create and modify tables, columns, and keys. For each subtype, the “Arc” flag must be set under “Table Mappings”.





Translation of Subtypes (6)

- Result of Method 1 ("Single Table Approach"):

  COMMITTEES(CNAME)
  COMMITTEES_FACULTY(COM_CNAME→COMMITTEES, INST_NAME→INSTRUCTORS)
  COURSES(CRN, TITLE, INST_NAME→INSTRUCTORS)
  INSTRUCTORS(ADDRESS^o, TENURED^o, NAME, EMAIL, INST_TYPE)

- The column "INST_TYPE" is declared to have values "EXT" and "FAC" (the short names of the subtypes).
- No CHECK-constraints are generated.
- The column sequence in INSTRUCTORS is strange.

Translation of Subtypes (7)

- Result of Method 2 ("Separate Table Approach"):

  COMMITTEES(CNAME)
  COMMITTEES_FACULTY(COM_CNAME→COMMITTEES, FAC_NAME→FACULTY)
  COURSES(CRN, TITLE, EXT_NAME^o→EXTERNAL, FAC_NAME^o→FACULTY)
  EXTERNAL(NAME, EMAIL, ADDRESS)
  FACULTY(NAME, EMAIL, TENURED)

- An arc is generated for the foreign keys in COURSES.
- The split table method for many-to-many relationships with the supertype ("AWARD1/2") is supported.

Translation of Subtypes (8)

- Result of "Implicit Sub-Type Approach":

  COMMITTEES(CNAME)
  COMMITTEES_FACULTY(COM_CNAME→COMMITTEES, FAC_NAME→FACULTY)
  COURSES(CRN, TITLE, INST_NAME^o→INSTRUCTORS, EXT_NAME^o→EXTERNAL, FAC_NAME^o→FACULTY)
  EXTERNAL(NAME, EMAIL, ADDRESS)
  FACULTY(NAME, EMAIL, TENURED)
  INSTRUCTORS(NAME, EMAIL)

- An arc is generated for the foreign keys in COURSES.
- This is Method 2 for partial specialization.

Translation of Subtypes (9)

- Result of Method 4 ("Arc Approach"):

  COMMITTEES(CNAME)
  COMMITTEES_FACULTY(COM_CNAME→COMMITTEES, FAC_1_FAC_ID→FACULTY)
  COURSES(CRN, TITLE, INST_NAME→INSTRUCTORS)
  EXTERNAL(ADDRESS, EXT_ID)
  FACULTY(TENURED, FAC_ID)
  INSTRUCTORS(NAME, EMAIL, EXT_EXT_ID^o→EXTERNAL, FAC_FAC_ID^o→FACULTY)

- The foreign keys in INSTRUCTORS are connected with an (optional) arc and marked as non-transferable.

Propagating Changes (1)

- It is probably best to start the DB Design Transformer only when one is finished with the ER-design.
- If one has already changed the relational schema, and then changes the ER-schema and runs the DB Design Transformer again, it is a difficult problem to merge both changes into one version.
- In general, it is important that the ER-Schema and the relational schema remain in sync — otherwise the ER-schema loses its value as a documentation for the created tables.

Propagating Changes (2)

- Of course, if one has not yet worked on the relational schema, one can simply delete it and run the DB Design Transformer again.

  Actually, it is not so simple to delete table definitions from the repository since they might be referenced in foreign keys. One must delete the foreign keys first. If one wants to delete all table definitions, one can click on the first, shift-click on the last, and then press the delete key. This will give an error message if a table is deleted that is still referenced by a foreign key. However, in Designer 6i (not Designer 6.0), one can choose to continue. After this is done, one simply presses "delete" again to remove the remaining tables (more runs might be needed, but if there are no cyclic foreign keys, finally all tables are deleted). In case of cyclic references, one must first delete at least one foreign key in the cycle before one can start to delete the tables.

Propagating Changes (3)

- Deleting the entire relational schema and running the DB Design Transformer again is the only completely automatic way that is guaranteed to keep both schemas in sync.
- The DB Design Transformer will never
  - ◇ remove existing tables (from a previous run) even if the corresponding entity type was deleted in the meantime,
  - ◇ remove columns from tables when the corresponding attribute was deleted.

Propagating Changes (4)

- The reason is probably that for denormalization, one could add columns and tables to the relational schema which are not present in the ER-schema.

  This should be a big exception, only if the performance requirements cannot be met with a good schema. But in earlier times it was done quite often (programmer time was cheap compared to hardware).

- The DB Design Transformer protects this work.

  The real reason probably is that in order to propagate deletions from the ER-schema to the relational schema, one must keep information about deleted schema elements. Also, the DB Design Transformer can be applied to a subset of the entity types. If one wants to delete tables, transforming the subset consisting of all entity types would be different from transforming the entire schema.

Propagating Changes (5)

- Under "Run Options" one can specify what the DB Design Transformer is allowed to modify.

  E.g. table names, column names, column sequence, column datatypes, etc.

- With the default (nothing can be modified) the DB Design Transformer remembers which elements in the ER-diagram are already mapped, and translates only new elements.

  E.g. if an attribute is added to an existing entity, it will be mapped to a new column in the existing table.

Propagating Changes (6)

- In the other extreme case (all modify options are checked), the new translation of the ER-schema overwrites the entire relational schema except that tables/columns are not deleted.
- E.g. even if one has renamed a column in the relational schema, running the DB Design Transformer again will reset it to its old name.

  I.e. the correspondence between ER-attributes and columns in tables is remembered in the repository, even if one of the two is renamed. One can see this information in the Repository Object Navigator under "`Usages/Implemented by Columns`" from the entity attribute.

Propagating Changes (7)

- One should not do arbitrary "last minute" changes in the relational schema. Go back to the ER-Schema and perform the required changes there!
- Depending on the kind of change, one can select the right modify options and run the DB Design Transformer only for the modified entity type.
- If something was deleted in the ER-schema, one must manually perform the corresponding deletion in the relational schema.

Overview

Design Editor (1)

- The relational schema generated by the database design transformer often still needs some work:
  - ◇ The names of the "intersection tables" for many-to-many relationships often must be changed.
  - ◇ Column names and the sequence of columns within a table might need changes.
  - ◇ Often, some constraints are missing.

    The DB Design transformer only generates keys, foreign keys, `NOT NULL`, and `CHECK` constraints for enumeration types or ranges automatically. Keys for one-to-one relationships are missing, as well as `CHECK`-constraints for subtypes, arcs, and other `CHECK`-constraints.

Design Editor (2)

- Manual work on the relational schema, continued:
  - ◇ For some foreign keys, one might have to select "`ON DELETE CASCADES`" etc.

    A default can be specified in the settings of the DB design transformer, but it might be useful to consider each case individually. E.g. for weak entities "`ON DELETE CASCADES`" is probably right.
- Warning: Many students submitted the result of the DB Design Transformer as logical schema without ever reading it. They all lost points.

  E.g. some generated column names are really ugly. Once application program development has started, it is difficult to change column or table names.

Design Editor (3)

- In addition, information necessary for the generation of application programs must be collected, e.g.
  - ◇ display title of the form generated for a table,
  - ◇ labels of input fields for columns,
  - ◇ field type (text, radio buttons, etc.),
  - ◇ field width,
  - ◇ help text,
  - ◇ display format (e.g. for date values),
  - ◇ columns that are not displayed.

Design Editor (4)

- After the logical design is finished, the following things must be defined:
  - ◇ Views.
  - ◇ Possibly triggers, stored procedures.
  - ◇ Users, table owners, access rights.
  - ◇ Physical design information.

    E.g. indexes, storage parameters for tables, distribution of tables over disks/tablespaces, etc. It is quite likely that the physical design will need to change when it turns out that the assumptions about the system load were not quite right. However, changing it after the data was loaded can be quite a lot of work.

Design Editor (5)

- Although this information can be edited directly with the Repository Object Navigator, Oracle offers a special tool for all this work: The Design Editor.
- The Design Editor consists of four distinct tools:
  - ◇ Server Model (Relational Database Schema)
  - ◇ Modules (Application Programs)
  - ◇ DB Administration (Users, Tablespaces, etc.)
  - ◇ Distribution (for Distributed Databases)

Design Editor (6)

- Later, first-cut application programs (for Oracle Developer Forms, Visual Basic, etc.) will be generated from the "module definitions".
- However, the module definitions contain only a link to the table name. The details such as the display width of input fields are defined in the server model (attached to tables).

  Of course, some things such as the exact position of the input fields on the form cannot be generated, and must be later edited with the programming tool itself (e.g. Oracle Developer Forms).

Design Editor (7)

- One window of the Design Editor is the Server Model Navigator.
- It looks very similar to the Repositor Object Navigator, but shows only objects that are part of the relational schema.

  A student thought that she could remove the relational schema (for a fresh run of the DB Design Transformer) by selecting the application system name at the top of the Server Model Navigator window and pressing "Delete". This removed her entire application system, not only the part shown in the window. For safety, export your design data at least once a day (with the Repository Object Navigator: "`Application`→`Export`") and copy them on a floppy disk.

Design Editor (8)

- The Design Editor uses normally wizards/tabbed dialog boxes instead of the simple property palette in the Repository Object Navigator.

  One can get also a property palette window under "`Tools`→`Property Palette`".
- The Design Editor also contains a tool to put information about the schema of an existing relational database in the repository.

  The "Design Capture Utility" ("`Generate`→`Capture Design of`→`Server Model`") can read the information from the data dictionary of an Oracle Database, from a file with SQL DDL (Create Table) commands, or via the ODBC interface.

Design Editor (9)

- The Design Editor has also a "Server Model Guide" which shows a tree of all server model object types:
  - ◇ Domains
  - ◇ Tables (Indexes, Triggers, Constraints)

    Constraints: Primary Keys, Foreign Keys, Unique Keys, Check.
  - ◇ Sequences
  - ◇ Advanced (Views, Snapshots, Clusters)
  - ◇ PL/SQL
  - ◇ Oracle8 (Collection Types, Object Types, Object Tables, Object Views).

Design Editor (10)

- When one selects an object type in the map, all objects of that type are shown. One can create, edit, or delete an object of the selected type.
- Basically, this is the same functionality as the "Server Model Navigator" which is also part of the Design Editor. Only the user interface is a bit different.

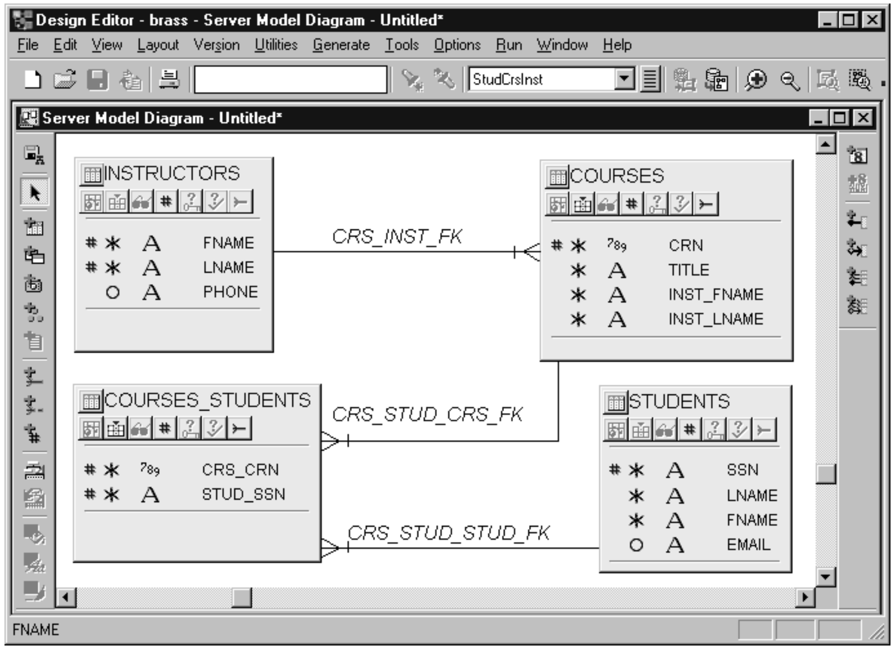
  One can also choose that the two tools are linked: When an object is selected in the Server Model Guide, it is automatically also selected in the Server Model Navigator. The Server Model Guide gives more advice what to do in which sequence and sometimes has links to documentation.

Server Model Diagrams (1)

- The "Server Model" part of the Design Editor has a graphical interface showing tables and their foreign key connections in "Server Model Diagrams".

  The easiest way to create a diagram is to expand the "Relational Table Definitions" in the "Server Model Navigator" on the left, then to select the tables that should appear on the diagram (e.g. click on the first table and shift-click on the last) and then to select "File→New→Server Model Diagram".
- These diagrams are quite similar to ER-Diagrams.

  However, the orientation on ER-diagrams is simpler. Server model diagrams are overloaded with information, table boxes are larger than entity boxes. Also many-to-many relationships are now shown as tables of their own, and foreign key columns do not appear on ER-diagrams.



Server Model Diagrams (3)

- Tables are shown as boxes with three sections:
  - ◇ The first section contains the table name and a number of buttons.

    Buttons: "Database Triggers", "Indexes", "Database Synonyms", "Primary Key", "Unique Keys", "Check Constraints", "Foreign Keys". A dimmed button means that the table has no object of that type. The button left to the table name is unusable.
  - ◇ The second section lists the columns (→ below).
  - ◇ The third section shows additional information as selected by the buttons in the first section.

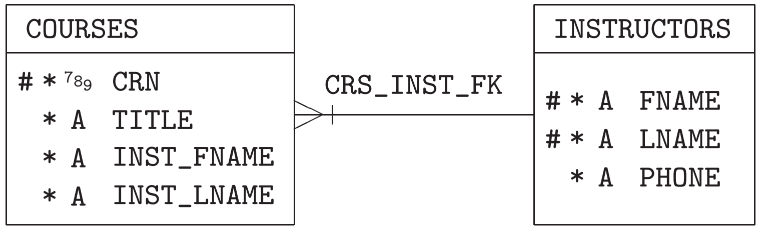
    If one selects "View→Track Associations" and clicks on e.g. an index, the corresponding columns are shown inverted above.

Server Model Diagrams (4)

- The second section of the table box contains one row per column with the following information:
  - ◇ "#": member of the primary key.
  - ◇ "*": mandatory column (not null),
    "○": optional column.
  - ◇ "▤": enumeration type value list.
  - ◇ "A": character/string data type,
    "789": numeric data type.
  - ◇ "123": sequence (unique number generator).
  - ◇ "▥": column belongs to domain.

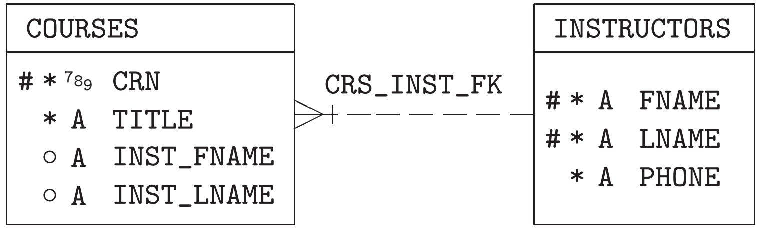
Server Model Diagrams (5)

- Foreign keys are shown as lines between the tables and use symbols similar to "one-to-many" relationships (but beware of the differences).
- Mandatory foreign keys (i.e. foreign keys that must be not null) are shown as solid lines:



Server Model Diagrams (6)

- Optional foreign keys (i.e. foreign keys that can be null) are shown as dashed lines:



- The entire line is either solid or dashed, there are no longer two halves.

Server Model Diagrams (7)

- Corresponding to the one-to-many relationship, the "crows foot" is on the side with the foreign key.

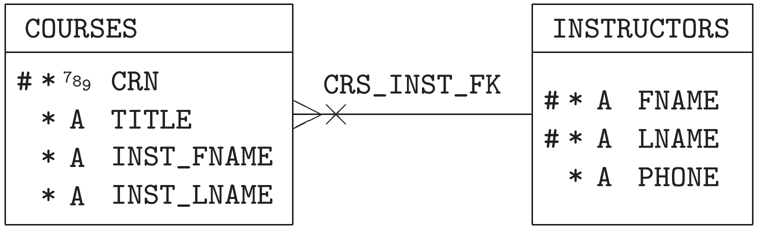
  It can also be seen as indicating the direction of the pointer, although a real arrowhead would be on the opposite side.

- The names of the foreign keys are often not helpful, but take space on the diagram.

  With "Options→Show/Hide" one can determine what is shown on the diagram. Removing the check mark from "Text" of "Associations" hides the foreign key names. One can specify which kinds of columns are shown, e.g. hide the foreign key columns on the diagram. One can also select which of the column type symbols are shown.

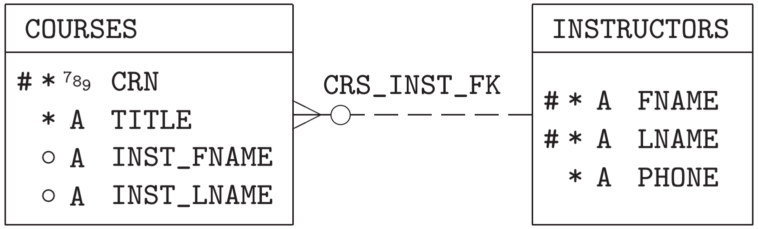
Server Model Diagrams (8)

- The small vertical bar near the crowsfoot means that deletions do not cascade ("restricted", one cannot delete an instructor that teaches courses).
- If one selects "ON DELETE CASCADES", the line is crossed with an "x":



Server Model Diagrams (9)

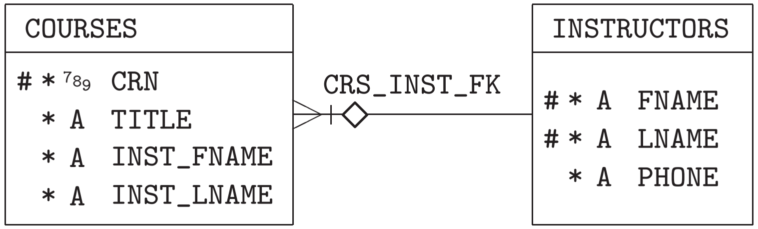
- If "ON DELETE SET NULL" is selected, a circle is used:



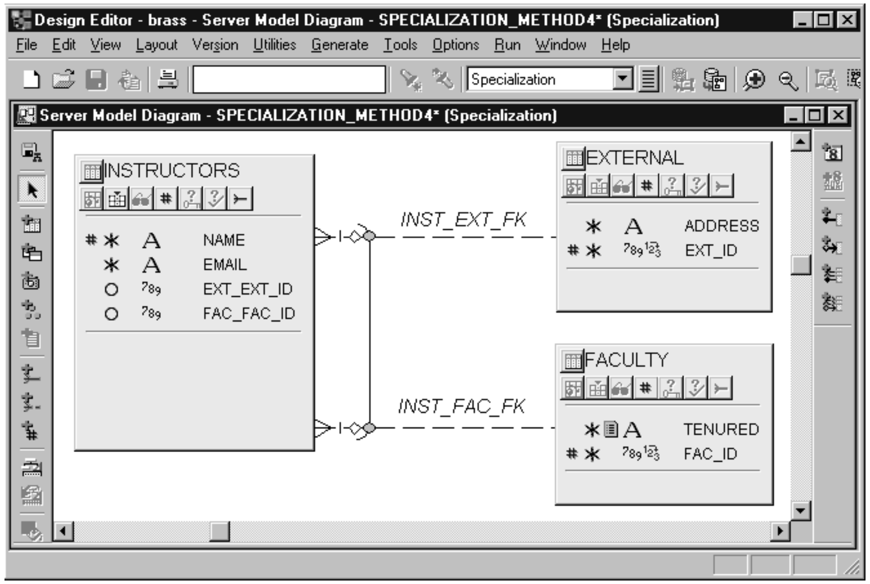
- A filled circle means "ON DELETE SET DEFAULT".
- The cascade rule for "ON UPDATE" is not shown on the diagram.

Server Model Diagrams (10)

- One can also mark foreign keys as non updatable (corresponding to a non-transferable relationship):



- Foreign keys can be marked as mutually exclusive by means of arcs (as on ER-diagrams).



Server Model Diagrams (12)

- The properties dialog box for tables (“Edit Table”) has tabs “Name”, “Columns”, “Display”, “Controls”, “UI”.

  Under “Display”, one can define which columns correspond to input fields in a form. Under “Controls” the type, size, etc. of these input fields is defined. Under “UI” (User Interface), more information about input fields is defined, e.g. a help text and a display format.

- Column and table names can be edited directly in the diagram, one does not have to go over the properties dialog box.

Server Model Diagrams (13)

- Tables also have the “Edit Text” dialog box, where one can define a description, notes, help text, and code for insert, update, delete, and locks.

  One can open this dialog box from the menu that appears if one right-clicks on the table. This menu also permits to add columns, triggers, indexes, synonyms, keys, check constraints, foreign keys.

- The properties dialog box for foreign keys has tabs “Foreign Key Mandatory”, “Foreign Key Column”, “Cascade Rules”, “Validation”.

  E.g. under “Validation” one can choose whether the constraint should be enforced on the server, the client, or both. One can also specify an error message and a table for exceptions (rows that violate it).

Server Model Diagrams (14)

- If one chooses to add e.g. a check constraint to a table, a wizard is opened that asks for the required information.

  To edit it later, display it in the third part of the box (by clicking on the button for the object type) and click on the symbol in front of the name. Clicking on the name only permits to edit the name. Editing an existing check constraint etc. shows the same screens as the wizard, but now one can jump with tabs between them.

- Oracle Designer does not check the SQL syntax e.g. of `CHECK`-constraint definitions.

  One can enter any text. Column names can be selected from a list. Of course, the exact SQL syntax depends on the DBMS.

Server Model Diagrams (15)

- The DB Design Transformer has already created indexes for foreign keys.

  In addition, the DBMS automatically creates indexes for primary and alternate keys.

- As part of the physical design, one can add further indexes to a table.
- One can also add triggers (e.g. for enforcing complex constraints or logging changes to a table).

Server Model Diagrams (16)

- Server model diagrams can also contain other objects, such as
  - ◇ Views (shown as grey-blue boxes).

    In order to create a view, one can e.g. right-click on the background of a server model diagram. Alternatively there is also a symbol on the left toolbar. A wizard is started that asks the required information. One can select base tables (`FROM`) and columns (`SELECT`), and then any `WHERE` clause can be entered.

  - ◇ Object types (shown as red boxes).

    In Oracle, an object type is a generalization of a record/row type. One can create one or more tables over an object type. Object type an tables are connected with a line that ends in a diamond attached to the table.

Server Model Diagrams (17)

- Objects on Server Model Diagrams, continued:
  - ◇ Clusters (shown as grey boxes).

    In Oracle, a cluster is a storage area in which rows of one or more tables may be stored, such that rows with the same value in the cluster column are stored together.

  - ◇ Snapshots (shown as light blue boxes).

    In Oracle, a snapshot is a copy of another table or view, used in distributed DBs for performance or failure safety reasons. One can specify that it is automatically refreshed at certain intervalls.

- The diagram legend can be shown in the upper left corner (it contains diagram title, author, date etc.).

Repository Reports

- Again, there are many repository reports which can be printed for documenting the DB Design, e.g.:
  - ◇ Entity to Table Implementation
  - ◇ Table Definition
  - ◇ Column Definition
  - ◇ Columns in Domain
  - ◇ Constraint Definition
  - ◇ Database Trigger
  - ◇ Cluster Definition
  - ◇ Tables, Columns, and Foreign Key Derivations

Overview

Database Administration (1)

- The following information can be specified with this part of the Design Editor:
  - ◇ Database Name and connection information.
  - ◇ Access information: Users, Roles, Profiles.
  - ◇ Storage Information: Tablespaces, Datafiles, Logfiles, Rollback Segments, Directories.
- In the Server Model view, the really physical information (like storage parameters) was not yet asked.

  Also, in the server model, the tables do not yet belong to users (there is no such property). One must move from the Server Model Relational Table Definitions to Table Implementations under "DB Admin".

Database Administration (2)

- One now can "implement" the tables under a user account in a database. A new wizard asks for a table from the server model and takes the designer through the physical options.

  One gets this wizard e.g. by selecting a user in the Database Administrator Guide, then selecting "Tables" and clicking on "Create". This does not mean that a table is created from scratch, one can select a table from the Server Model. Since often several tables have the same storage parameters, one can create named sets of such parameters ("Storage Definitions") and assign to tables.
- Of course, the resulting data are still stored in the repository. The table is not yet really implemented.

Database Administration (3)

- The Database Administration part of the Design Editor is only a subset of the Repository Object Navigator (there are no new diagrams).

  However, again wizards/tabbed dialog boxes are used instead of the property palette. And it has a "Database Administrator Guide" that shows the steps for specifying a database.
- The tool is similar to a graphical user interface for a DBA (but stores all information in the repository).
- From the collected information, database creation scripts can be generated.

Overview

1. Database Design Transformer
2. Design Editor: Server Model Diagrams
3. Design Editor: Database Administration
4. Generation of SQL Code

DDL Generation (1)

- The Database Design Transformer stores the relational schema in the repository. It does not actually create the tables.
- The reason for this is that in most cases, some things must still be changed/added manually.
- Once one is satisfied with the relational schema, one can generate SQL DDL code containing e.g. `CREATE TABLE` statements.

  DDL = Data Definition Language. The generation is done with the Design Editor: "`Generate→Generate Database from Server Model`".

DDL Generation (2)

- Oracle Designer can create DDL code for different DBMS: ANSI 92, DB2, Oracle (different versions), RDB7, SQL Server, Sybase.
- The creation of tables etc. can be done as follows:
  - ◇ Files with DDL statements are created, these must be executed manually in the target DB.
  - ◇ If the target database is an Oracle Database, Oracle Designer can directly create the tables.
  - ◇ If the target DB supports ODBC connections, tables can also be directly created.

DDL Generation (3)

- One can select for which schema objects should be generated (e.g. only a subset of the tables).

  This is done on the "`Objects`" tab. E.g. click on the double right arrow: "`Generate All`".

- What can be generated, depends on the DBMS chosen, e.g.:
  - ◇ "`ANSI 92`": Only tables and views.
  - ◇ "`SQL Server`" Only domains, tables, and views.

    Which is strange, since it has indexes.

DDL Generation (4)

- When creating files, one defines a file prefix (e.g. `courses`) and a directory. The different kinds of schema elements will then be written to different files (for Oracle8):
  - ◇ `courses.tab`: Table Definitions
  - ◇ `courses.con`: Constraints (as `ALTER TABLE ...`)
  - ◇ `courses.ind`: Indexes
  - ◇ `courses.sqs`: Sequence Definitions
  - ◇ `courses.sql`: Includes all of the above files.