

A technical discussion of UML
06/11/03

Rational. software



Relational Modeling with UML

Davor Gornik

Table of Contents

Relational Databases	1
Core Elements of Relational Modeling.....	1
Tables.....	1
Relationships	2
Extended Relational Modeling	4
Views.....	4
Stored Procedure.....	5
Constraints.....	6
Triggers	7
ER Models and Relational Modeling	9
Summary.....	12

Relational Databases

Relational databases are the most commonly used type of data storage. This is due, in large part, to the fact that the simplicity of the storage principles offers users greater flexibility. Also, the table-like structures map easily to real-life data formats such as forms and spreadsheets.

Entity Relationship (ER) modeling defines the methodology used in the analysis and design of information-based systems. The output is a list of entity types, relationship types, and constraints. ER modeling is based on artifacts, which can be either a representation of physical artifacts, such as Product or Employee, or a representation of a transaction between artifacts, such as Order or Delivery.

A relational model can be directly transformed into database structures. The model includes all of the essential elements of the relational concept like tables and relationships, as well as enhanced capabilities of modern databases like tablespaces and indexes. Relational modeling describes the design of artifact-based systems with relational databases.

Relational modeling is used as an abstraction of the database implementation. Database designers are the primary users of these models. Analysts, developers, testers, and other team members of a software development team are the secondary audience. As such, it is important to make the relational models understandable for everyone to prevent mistakes in the creation and use of the database.

This white paper explains the basic concepts behind relational modeling, how it applies to databases, and how it is implemented using the Unified Modeling Language (UML). In addition, it identifies differences between ER modeling and relational modeling. While ER modeling suits the analysis of the problem, relational modeling deals with the implementation.

Core Elements of Relational Modeling

The relational model builds upon the basic concept of a table. The table is built from tuples of identical structure called rows. Each row contains data assigned to columns.

The relational model assumes that every tuple has a unique identifier based on the data stored in it. The unique identifier can be a single column or a combination of columns. However the table requires that all rows use the same columns or a combination of columns as a unique identifier. The unique identifier can be used by other tuples in the same or a different table as the address to the tuple.

Tuples that address other tuples specify a reference in the format of column values corresponding to the identifier. The columns needed for the reference must be added to the structure of the tuple, and therefore, to the structure of the table.

Two essential elements of relational modeling are tables and relationships.

Tables

The element containing tuples of the same structure is called a table. It is the core building block of the relational model. The tuples are called rows and contain columns. Each column defines the possible content using a data type that can be stored in it.

The following table represents information on cars. The table has five rows of data describing five different cars. Each car is described in one row with five columns. The columns are LicenseNumber, Make, Model, Year, and Color. The LicenseNumber is the way to uniquely identify a particular tuple. It is the primary key of the column. The rows in the table are not ordered.

LicenseNumber	Brand	Model	Year	Color
A17046	BMW	X5	2001	black
A13921	HONDA	ODYSSEY	2002	gold
A13502	VOLVO	S60	2002	silver
A18342	PEUGEOT	406	2000	red
A15022	INFINITY	Q45	2002	silver

Reading just the titles of columns identifies the structure of a table.

The structure of a table specifies the columns of the table and vice versa. Columns are identified by a name, which has to be unique in a table. An optional comment defines the meaning of the content of a column in human language. A data type tells the database how to handle the column and what kind of data will be stored in the column.

The null ability expresses if data in every row of a column is required or optional. We can see it only explicitly in columns where values might be missing.

LicenseNumber	Brand	Model	Year	Color
---------------	-------	-------	------	-------

In UML, each table is identified by a name and displayed in a rectangle. We chose the name Cars for our table. The name is displayed in the first compartment together with the stereotype ¹that identifies the rectangle as a table. With UML, stereotype defines a specialization of a common construct and extends the semantics. In this case the stereotype table defines that the element describes a structure of a table.

The columns are defined in the second compartment of the rectangular shape. Each column is specified by a name followed by a colon (:) and the data type. Additional symbols in front of the name define the null ability of each column. The primary key is identified with a “PK.”

In the table below, the LicenseNumber is the primary key of the table Cars. The Brand and Year are not optional, and data must be provided for these columns in every row in the table Cars. The Model and Color are optional and data for these columns is not required in every row.

<p><<table>> Cars</p>
<p>PK LicenseNumber : CHAR(8) NN Brand : VARCHAR(20) N Model : VARCHAR(20) NN Year : SMALLINT N Color : VARCHAR(20)</p>

Relationships

It is nice to know which cars exist, but we would also like to know who owns these cars. The Residents table specifies potential owners of cars. In order to show car ownership, we must establish a relationship between Residents and Cars. The primary key of the Residents table is the SocialSecurityNumber.

SocialSecurityNumber	FirstName	MiddleName	LastName	DateOfBirth	Sex
232483654	Jim	Evans	Bellevue	7/13/1971	M
254865482	Eva	Weber	Adams	8/16/1974	F
865947523	Ken		Dietrich	2/3/1952	M
859647521	Barbara		Mc Kenna	11/2/1951	F

¹ Stereotypes define a new type of artifact that extends the semantics of the metamodel. For example the stereotype <<table>> defines a special type of artifact class with columns that have public access.

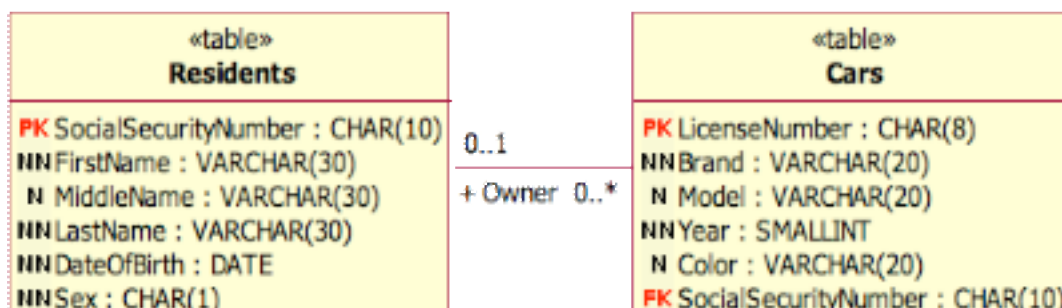
Cars and Residents are independent of each other. The existence of a resident is not required for the existence of a car. Additionally, although not every resident may have a car, one resident can own a number of cars. In this scenario, the Residents are the parent table in the relationship, while Cars is the child table. The relationship is established by including the identification columns of the parent table in the child table. In addition, the SocialSecurityNumber is added to the table Cars because of the relationship to the Residents table.

LicenseNumber	Make	Model	Year	Color	SocialSecurityNumber
A17046	BMW	X5	2001	black	865947523
A13921	HONDA	ODYSSEY	2002	gold	254865482
A13502	VOLVO	S60	2002	silver	865947523
A18342	PEUGEOT	406	2000	red	232483654
A15022	INFINITY	Q45	2002	silver	254865482

The table definition contains not only the data in the column, but also the link to where the source of the column is – the Residents table. The link is defined with a relationship.

The modeling of a non-identifying relationship between independent tables Residents and Cars connects the tables with a straight line. The line is accompanied by numbers that specify the cardinality of each table in the participating relationship.

In the example below, the numbers define that a resident can be the owner of any number of cars, including none, while each car can only be owned by one resident. The text on the end of a relationship defines the role of the participant in the relationship. The resident acts in the role of the owner of the car. Another possible role might be a driver, for example.



Foreign keys in a table are columns that directly relate to columns in the parent table of a relationship.

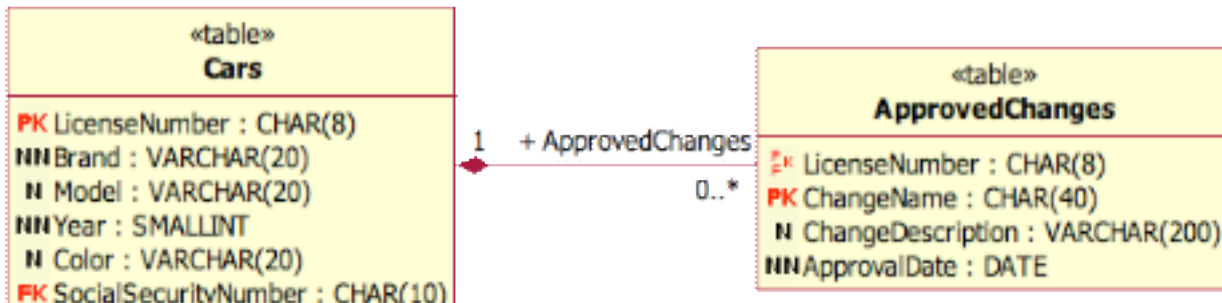
Cars can have approved changes. For example, the exhaust system might be changed or a replacement made of the complete engine. Approved changes are represented in a separate table named ApprovedChanges.

ChangeName	ChangeDescription	ApprovalDate
Sports Exhaust	Brand Loud Devil, Model Polished Tiger	3/19/2002
New Engine	New Serial No: 458D093847233-0231	8/11/2002
New Engine	New Serial No: 93820XJ03988742BDD	2/1/2003

The column ChangeName is not enough to specify the uniqueness of a row. Therefore, the table does not have an independent primary key. It is completely dependent on cars since the changes depend on a specific car. In such cases we add the primary key of the parent table to the child table and combine the parent primary key with additional columns of the child table (if needed) to form the primary key of the child table.

LicenseNumber	ChangeName	ChangeDescription	ApprovalDate
A17046	Sports Exhaust	Brand Loud Devil, Model Polished Tiger	3/19/2002
A13921	New Engine	New Serial No: 458D093847233-0231	8/11/2002
A13502	New Engine	New Serial No: 93820XJ03988742BDD	2/1/2003

The joined columns LicenseNumber and ChangeName form the primary key. The LicenseNumber is needed because of the complete dependency to the car. This kind of dependency is called an identifying relationship. The identifying relationship has to be built between two tables.



The identifying relationship adds another primary key column to the child table. Each row in ApprovedChanges must have an associated car. Each car can have any number of approved changes, including none. The role name from the relationship can be omitted.

Extended Relational Modeling

Over the years, vendors have expanded the concepts behind relational modeling by developing very powerful databases that enhance the usability of data with the introduction of virtual tables, constraints, and functional components that ensure consistency in the database.

Views

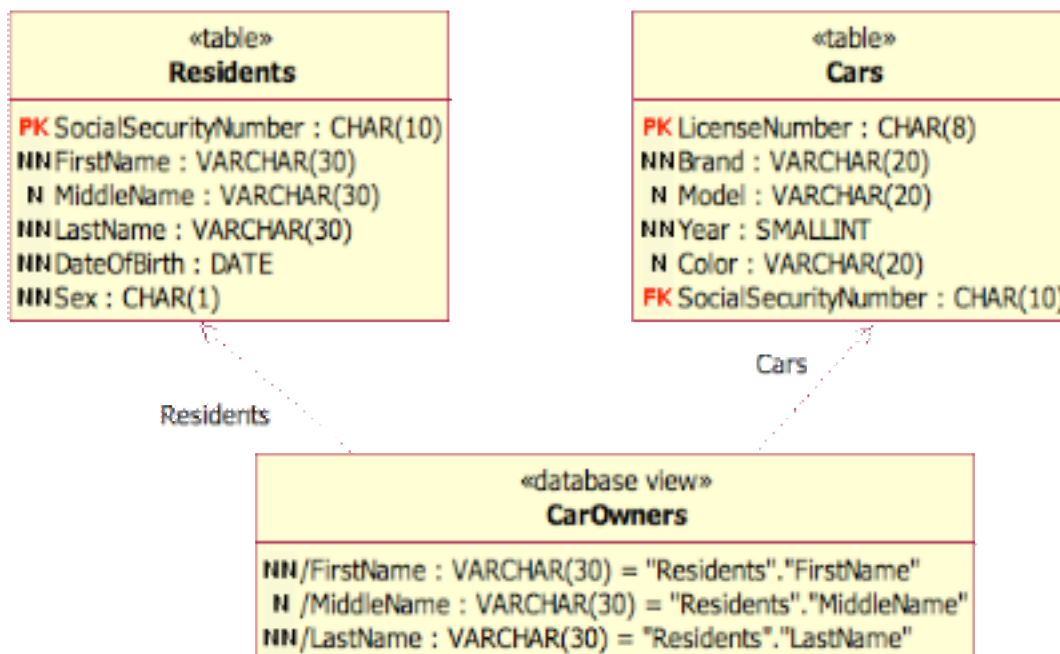
A table represents the physical structure of data in the database. Often we need different views on data that are filtered, contain a smaller number of rows, or include related data from several tables.

For example, if we wanted to discover the brand of cars residents own, we would generate a view on the data. The view would create a virtual table with the owner and the associated car brand. The Cars and Residents tables have to be joined for this reason. This join is described in the relationship.

FirstName	MiddleName	LastName	Make
Jim	Evans	Bellevue	PEUGEOT
Eva	Weber	Adams	HONDA
Eva	Weber	Adams	INFINITY
Ken		Dietrich	BMW
Ken		Dietrich	VOLVO

The structure of the view contains three essential components: the source of data, the structure of the data in the view, and the transformation information – often referred to as selection. The source for the data is the tables Residents and Cars. The structure of the data is the four columns displayed in the view: FirstName, MiddleName, LastName, and Make. The transformation information describes how the rows in the view are built from the source. This view joins the rows of residents with rows of cars wherever there is a match between the car and the resident in the SocialSecurityNumber field.

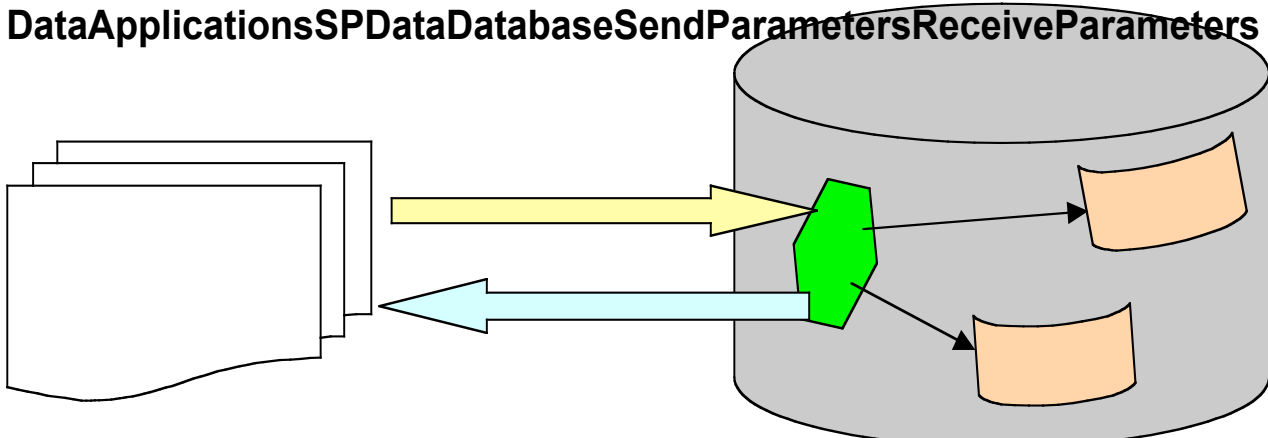
In UML, a view is displayed as a rectangle with a stereotype of <<database view>> together with the name. The source for the data is identified with dependencies, which are directed dashed connectors to tables or other views. The dependencies can be named for better understanding in bigger diagrams. The structure of the view is described in the second compartment of the rectangle representing it. The information about the structure is actually derived from the source (when possible or computed) and is displayed in the compartment for clarification. The information includes the null ability, the name of the column that can differ from the source column, the data type, and the source of the data in the column.



Stored Procedure

Storing data is just one of the functionalities of a database. The database has to offer a way to store, retrieve, evaluate, and update data. Users execute transformations or business logic to ensure consistency of the data. Often users have to execute the same operation over and over again. Instead of executing the code dynamically with each request, the code can be stored in the database itself. Specified parameters allow necessary variability for the execution like the customer name, order number, etc.

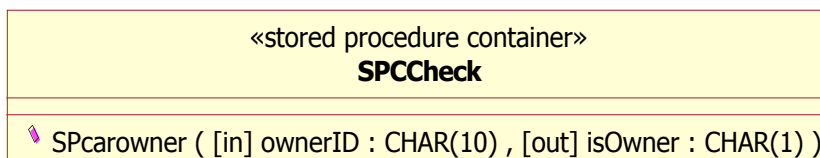
DataApplicationsSPDataDatabaseSendParametersReceiveParameters



The application does not access the data directly. Stored procedures hide the data structure and the relationship between data constructs. A call to the stored procedure supplies parameters that are used to execute the stored business logic. The stored procedure acts as a black box for the user. It might access any data in the database. Stored procedures have very stable interfaces that allow changes in the physical data structure without the disconnection with the applications.

Stored procedures are grouped into containers during the design phase to have a better overview of the interface. The database might have an implementation for the container or not, depending on the database type. In UML, the stored procedure container is represented by a rectangle with three compartments. The first compartment contains the name of the stored procedure container and the stereotype `<<stored procedure container>>`. Stored procedures are normally grouped into a container based on their functionality. The example below shows the container `SPCCheck` that will contain all of the stored procedures that check a status of data in the database.

The third compartment contains the name of the stored procedure and a list of parameters. There are two types of parameters: parameters that are sent for the execution of the stored procedure and parameters that are returned as results from the stored procedure. Additionally, a parameter can be submitted with the call and modified by the stored procedure. The modifiers in the signature of the stored procedure are `[in]` for the parameters that are sent with the call of the stored procedure, `[out]` for results, and `[inout]` for parameters that are sent with the call and can be modified by the stored procedure.



Functions are a special type of stored procedure. Functions are typed and return implicitly a parameter as the function name. The parameter that is returned is identified by the modifier `[return]`.

Constraints

The structure of the database does not provide any enforcement for the data. For example, users can provide data that is not planned (status 'UNKNOWN'), an incorrect combination of data (date of birth later than date of employment), and non-supported numbers (number of cars owned is -2). Constraints are used to prohibit such occurrences of wrong data.

In the example below, we create a constraint that limits the approval dates to dates after January 1, 1990. Every operation on data in the table `ApprovedChanges` will implicitly check the constraint.

LicenseNumber	ChangeName	ChangeDescription	ApprovalDate
			>= 1/1/1990
A17046	Sports Exhaust	Brand Loud Devil, Model Polished Tiger	3/19/2002
A13921	New Engine	New Serial No: 458D093847233-0231	8/11/2002
A13502	New Engine	New Serial No: 93820XJ03988742BDD	2/1/2003

Relationships are also considered constraints. They add either primary key or foreign keys to the specification of a table. The constraint of a primary key requires that data stored in all key combinations be unique. The constraint for a foreign key defines that data in the column combination selected as a reference has to correspond to the data of the column combination of the parent table.

Constraints are not independent. They restrict a certain data structure. Constraints are shown in the third compartment of a table.

«table» ApprovedChanges	
PK	«column» LicenseNumber : CHAR(8)
PK	«column» ChangeName : CHAR(40)
N	«column» ChangeDescription : VARCHAR(200)
NN	«column» ApprovalDate : DATE
	«primary key» PKApprovedChanges ()
	«foreign key» FKCars ()
	«check constraint» CCApprovalDate ()

A stereotype defines the type of constraint. Constraints do not have parameters. They are executed implicitly. The parentheses following the name of the constraint imply code execution.

The foreign key constraints cannot exist without a valid relationship to another table. The specification of the foreign key constraint needs the complete definition of a relationship, including the parent table, the constraint or columns referenced in the parent table, and the columns used in the client table.

The primary key constraint requires columns associated to the primary key. The check constraint relates to a single column or a combination of columns as defined by the code of the constraint.

Triggers

Constraints can either allow or prohibit activities in the database. Triggers take active action when certain events occur. These events could be any activities that change the content of a database, including deletion, insertion of new data, and change of existing data. For example, we could specify a trigger that would automatically change the ownership of cars when a record for a resident is deleted.

Here is the snapshot of data before Ken Dietrich (SocialSecurityNumber 865947523) is removed from the database.

SocialSecurityNumber	FirstName	MiddleName	LastName	DateOfBirth	Sex
232483654	Jim	Evans	Bellevue	7/13/1971	M
254865482	Eva	Weber	Adams	8/16/1974	F
865947523	Ken		Dietrich	2/3/1952	M
859647521	Barbara		Mc Kenna	11/2/1951	F

Two cars are owned by Ken before his record is removed from the database. These are the BMW (LicenseNumber A17046) and VOLVO (LicenseNumber A13502).

LicenseNumber	Make	Model	Year	Color	SocialSecurityNumber
A17046	BMW	X5	2001	black	865947523
A13921	HONDA	ODYSSEY	2002	gold	254865482
A13502	VOLVO	S60	2002	silver	865947523
A18342	PEUGEOT	406	2000	red	232483654
A15022	INFINITY	Q45	2002	silver	254865482

The trigger could automatically start just before the record is deleted and remove the ownership from the cars.



SocialSecurityNumber	FirstName	MiddleName	LastName	DateOfBirth	Sex
232483654	Jim	Evans	Bellevue	7/13/1971	M
254865482	Eva	Weber	Adams	8/16/1974	F
865947523	Ken		Dietrich	2/3/1952	M
859647521	Barbara		Mc Kenna	11/2/1951	F

The activity of deletion of the record starts the trigger, which executes the assigned code (in this case, setting the SocialSecurityNumber to NULL).

LicenseNumber	Make	Model	Year	Color	SocialSecurityNumber
A17046	BMW	X5	2001	black	NULL
A13921	HONDA	ODYSSEY	2002	gold	254865482
A13502	VOLVO	S60	2002	silver	NULL
A18342	PEUGEOT	406	2000	red	232483654
A15022	INFINITY	Q45	2002	silver	254865482

Each trigger is specified according to the event that starts it (update, delete, insert) and is assigned to a table. The trigger is displayed in the third compartment of a table with the stereotype <<trigger>> followed by a name. Triggers do not have explicit parameters; they use the value of the row that is changed if required by the code of the trigger.

In the example below, the trigger event is displayed with the respective icon (D for delete, U for update, I for insert).

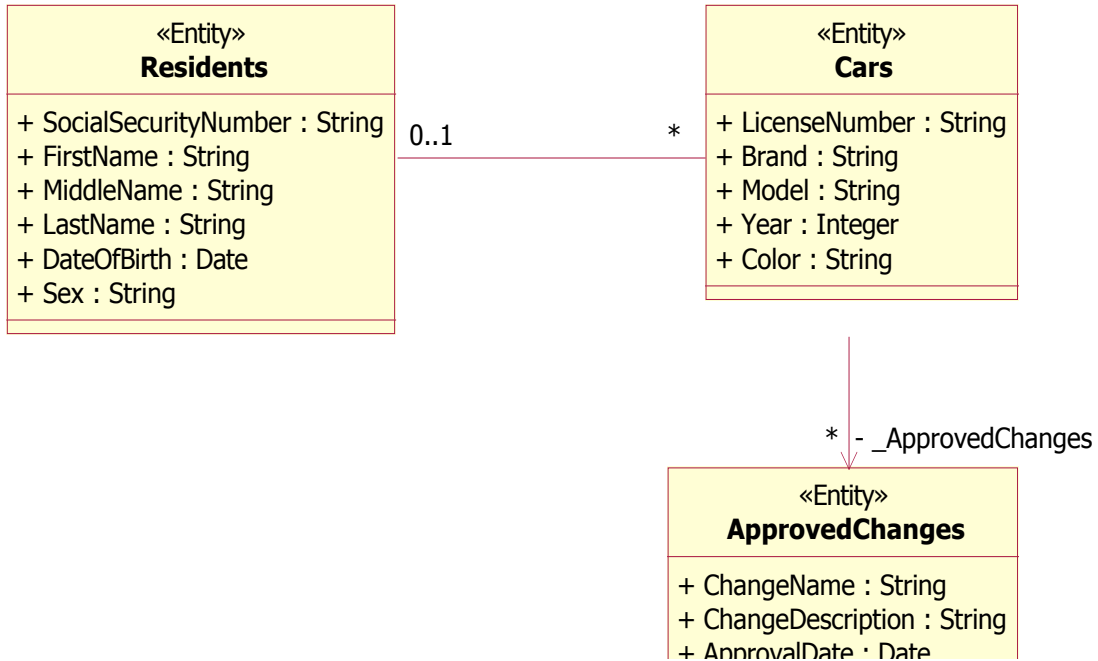
«table» Residents	
PK	«column» SocialSecurityNumber : CHAR(10)
NN	«column» FirstName : VARCHAR(30)
N	«column» MiddleName : VARCHAR(30)
NN	«column» LastName : VARCHAR(30)
NN	«column» DateOfBirth : DATE
NN	«column» Sex : CHAR(1)
	«primary key» PKResidents ()
	«trigger» TGDelete ()

ER Models and Relational Modeling

ER models specify entities and relationships between entities. They are not able to cover all of the extended relational modeling because they do not implement constructs like triggers and stored procedures. The flexibility of relationships is much bigger in the ER model than it is in the relational model, which is limited to only two types of relationships.

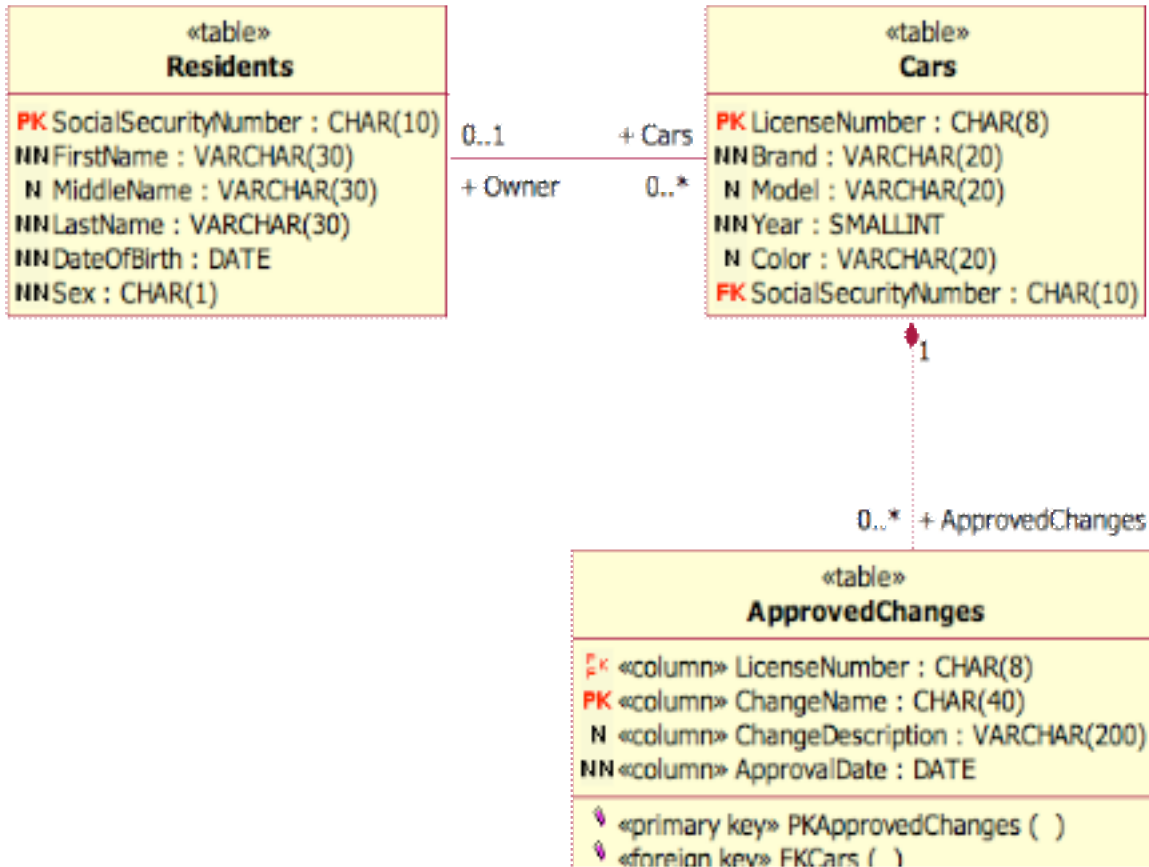
However, ER models can be transformed into a subset of relational models.

In the example we used earlier regarding residents, cars, and approved changes, the ER model served as a source for the relational model. The Residents and the Cars entities are associated to each other. The entity Residents is associated with any number of Cars. Each Car is associated with 0 or 1 Residents.



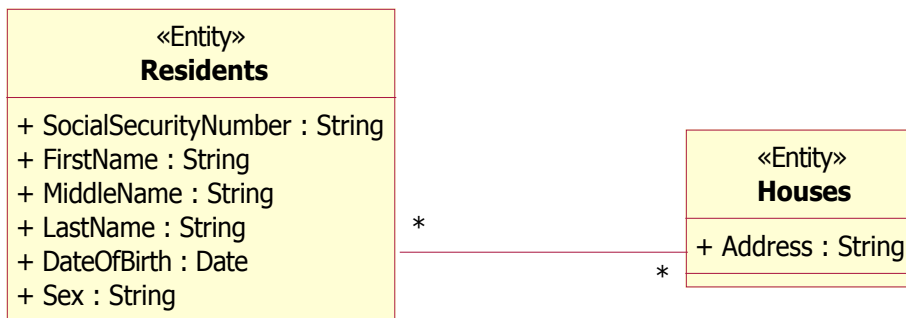
An aggregation specifies the relationship between **Cars** and **ApprovedChanges**. The cardinality associated with the aggregation is unlimited, which means that every car could have any number of approved changes.

The data types associated with attributes of entities are not database dependent. In this example, it was agreed to use universal data types like String, Date, and Integer. These data types convert to database specific data types.

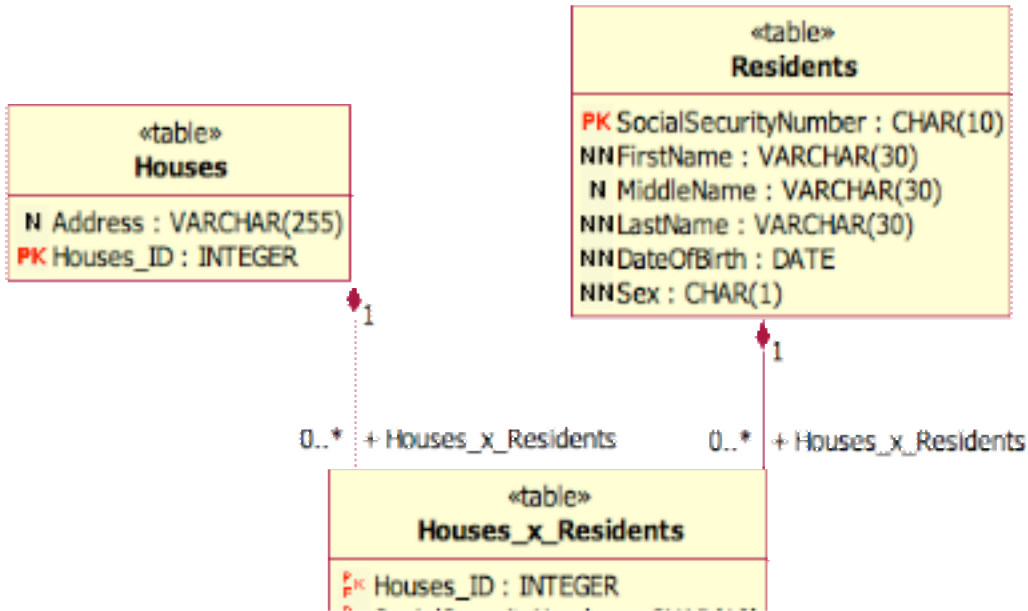


The transformed relational representation using UML employs almost the same syntax. The representation adds additional information needed for the implementation of the relational database.

The transformation can also perform more complex operations. Many-to-many associations do not have a parent and child entity. Both entities are completely independent and it is impossible to decide which would play the role of a parent in the key migration. Therefore, transformation of a many-to-many association results in a new table.



The table represents the complexity of the association. It has two identifying relationships to the parent tables. The identifying association migrates the primary keys to the new table where they are both part of the unique identifier.



In the diagram representing the relational model, we did not display constraints. UML gives you the freedom of display formats that best suit the need. In this case, the most important are the relationships themselves.

Triggers do not have a corresponding representation in the ER models. They are a construct used in the development of databases.

Stored procedures are procedural interface to the database. They are independent of entities. Therefore, stored procedures also do not find a representation in the ER model.

Summary

Relational models are the first abstraction of a relational database. They follow the same constraints as relational database. We use the models to explain the database structure to a broad audience, including DBA, database designers, analysts, developers, and testers.

UML is a common language used by the analyst, developer, and tester community. This paper discussed usage of the UML for relational models to increase understanding of the database in the complete development team.

To learn more about ER modeling and how UML applies to ER modeling, please see the paper “Entity Relationship Modeling with UML.”



IBM software integrated solutions

IBM Rational supports a wealth of other offerings from IBM software. IBM software solutions can give you the power to achieve your priority business and IT goals.

- *DB2[®] software helps you leverage information with solutions for data enablement, data management, and data distribution.*
- *Lotus[®] software helps your staff be productive with solutions for authoring, managing, communicating, and sharing knowledge.*
- *Tivoli[®] software helps you manage the technology that runs your e-business infrastructure.*
- *WebSphere[®] software helps you extend your existing business-critical processes to the Web.*
- *Rational[®] software helps you improve your software development capability with tools, services, and best practices.*

Rational software from IBM

Rational software from IBM helps organizations create business value by improving their software development capability. The Rational software development platform integrates software engineering best practices, tools, and services. With it, organizations thrive in an on demand world by being more responsive, resilient, and focused. Rational's standards-based, cross-platform solution helps software development teams create and extend business applications, embedded systems and software products. Ninety-eight of the Fortune 100 rely on Rational tools to build better software, faster. Additional information is available at www.rational.com and www.therationaledge.com, the monthly e-zine for the Rational community.

IBM is a wholly owned subsidiary of the IBM Corporation. (c) Copyright Rational Software Corporation, 2003. All rights reserved.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Printed in the United States of America
05-03 All Rights Reserved. Made in the
U.S.A.

IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Rational, and the Rational logo are trademarks or registered trademarks of Rational Software Corporation in the United States, other countries or both.

Other company, product or service names may be trademarks or service marks of others.

The IBM home page on the Internet can be found at **ibm.com**