

Attribute = Column AND Row = Tuple

INTRODUCTION TO DATABASES USING ORACLE 420-983-VA

RELATIONAL MODEL

WHAT IS A DATA MODEL?

A **data model** is a notation for describing data or information. The description generally consists of three parts:

1. **Structure of the data.** You may be familiar with tools in programming languages such as C or Java for describing the structure of the data used by a program: arrays and structures or objects, for example. In the database world, data models are at a somewhat higher level than data structures, and are sometimes referred to as a conceptual model to emphasize the difference in level.
2. **Operations on the data.** In programming languages, operations on the data are generally anything that can be programmed. In database data models, there is usually a limited set of operations that can be performed. We are generally allowed to perform a limited set of queries (operations that retrieve information) and modifications (operations that change the database). This limitation is not a weakness, but a strength. By limiting operations, it is possible for programmers to describe database operations at a very high level, yet have the database management system implement the operations efficiently.
3. **Constraints on the data.** Database data models usually have a way to describe limitations on what the data can be. These constraints can range from the simple (e.g., “a day of the week is an integer between 1 and 7” or “a movie has at most one title”) to some very complex limitations.

THE RELATIONAL MODEL

Codd proposed the relational data model in 1970. At that time, most database systems were based on one of two older data models: the hierarchical model and the network model. The relational model revolutionized the database field and largely supplanted these earlier models.

The relational model is very simple and elegant: **a database is a collection of one or more relations, where each relation is a two-dimensional table with rows and columns.**

This simple tabular representation enables even novice users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data. The major advantages of the relational model over the older data models are its simple data representation and the ease with which even complex queries can be expressed.

THE RELATIONAL MODEL

Example of a relation

Each column represent a property of movies

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

Each row represent a movie

The diagram illustrates a table representing a relation. The table has four columns: 'title', 'year', 'length', and 'genre'. The first row contains the values 'Gone with the wind', '1939', '231', and 'drama'. The second row contains 'Star Wars', '1977', '124', and 'sci-fi'. The third row contains 'Wayne's World', '1992', '95', and 'comedy'. Red arrows point from the text 'Each column represent a property of movies' to the column headers. Red arrows point from the text 'Each row represent a movie' to the data rows.

THE RELATIONAL MODEL

Attributes

The columns of a relation are named by **attributes** (also called fields or columns). Attributes appear at the tops of the columns. Usually, an attribute describes the meaning of entries in the column below. For instance, the column with attribute length holds the length, in minutes, of each movie.

In our Movies relation example the attributes are: “title”, “year”, “length”, “genre”

Attributes (Columns, Fields)

Attribute Name

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Schemas

The name of a relation and the set of attributes for a relation and the domain of each attribute is called the **schema** for that relation. We show the schema for the relation with the relation name followed by a parenthesized list of its attributes.

Schema for the Movies relation:

```
Movies(title, year, length, genre)
```

The attributes in a relation schema are a set, not a list. However, in order to talk about relations we often must specify a “standard” order for the attributes.

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Schemas

In the relational model, a database consists of one or more relations. The set of schemas for the relations of a database is called a [relational database schema](#), or just a [database schema](#).

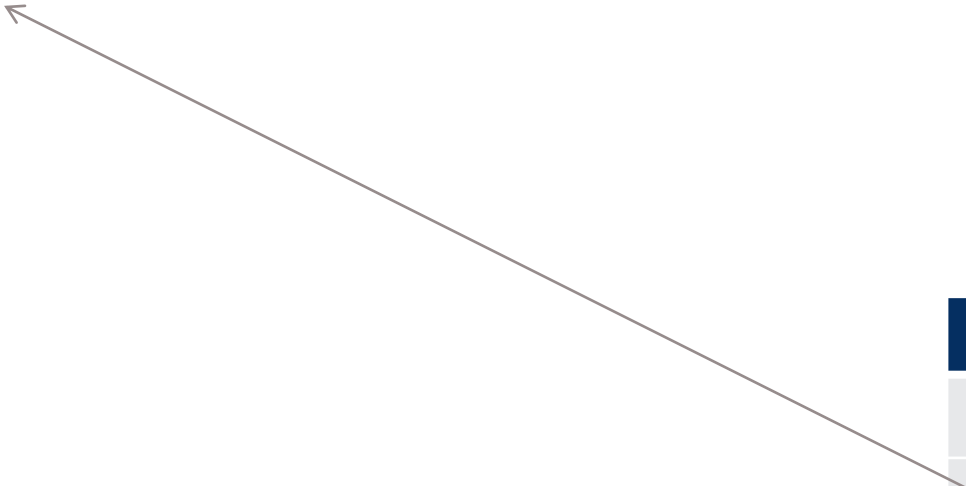
title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Tuples (records)

The rows of a relation, other than the header row containing the attribute names, are called **tuples** (also called **records** or **rows**). A tuple has one component for each attribute of the relation.

For example, the second tuple from the Movie relation has the four components: “Star Wars”, 1977, 124 and “sci-fi” for the attributes **title**, **year**, **length** and **genre**.



title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Domains

The relational model requires that each component of each tuple be atomic; that is, it must be of some elementary type such as integer or string. It is not permitted for a value to be a record structure, set, list, array, or any other type that reasonably can have its values broken into smaller components.

It is assumed that associated with each attribute of a relation is a domain, that is, a particular elementary type. The components of any tuple of the relation must have, in each component, a value that belongs to the domain of the corresponding column.

It is possible to include the domain, or data type, for each attribute in a relation schema. We shall do so by appending a colon and a type after attributes. For example, we could represent the schema for the Movies relation as:

Movies(title: string, year: integer, length: integer, genre: integer)

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Equivalent Representations of a Relation

Relations are sets of tuples, not lists of tuples. Thus the order in which the tuples of a relation are presented is immaterial. Moreover, we can reorder the attributes of the relation as we choose, without changing the relation.

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy



title	genre	year	length
Gone with the wind	drama	1939	231
Star Wars	sci-fi	1977	124
Wayne's World	comedy	1992	95

THE RELATIONAL MODEL

Relation Instances

A relation about movies is not static; rather, relations change over time. We expect to insert tuples for new movies, as these appear. We also expect changes to existing tuples if we get revised or corrected information about a movie, and perhaps deletion of tuples for movies that are expelled from the database for some reason.

It is less common for the schema of a relation to change. However, there are situations where we might want to add or delete attributes. Schema changes, while possible in commercial database systems, can be very expensive, because each of perhaps millions of tuples needs to be rewritten to add or delete components.

We call a set of tuples for a given relation an **instance** of that relation.

(Current) Instance of relation Movies

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy


THE RELATIONAL MODEL

Keys

There are many constraints on relations that the relational model allows us to place on database schemas. One kind of constraint is so fundamental that we shall introduce it here: key constraints. A set of attributes forms a key for a relation if we do not allow two tuples in a relation instance to have the same values in all the attributes of the key.

We indicate the attribute or attributes that form a key for a relation by underlining the key attribute(s). For instance, the Movies relation could have its schema written as:

Movies(title, year, length, genre)



title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Keys

A set of attributes forms a key for a relation is a statement about all possible instances of the relation, not a statement about a single instance. For example, looking only at our Movie relation instance, we might imagine that genre by itself forms a key, since we do not see two tuples that agree on the value of their genre components. However, we can easily imagine that if the relation instance contained more movies, there would be many dramas, many comedies, and so on.

title	year	length	genre
Gone with the wind	1939	231	drama
Star Wars	1977	124	sci-fi
Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Artificial Keys

While we might be sure that title and year can serve as a key for Movies, many real-world databases use artificial keys, doubting that it is safe to make any assumption about the values of attributes outside their control. For example, companies generally assign employee ID's to all employees, and these ID's are carefully chosen to be unique numbers.

ID	title	year	length	genre
101	Gone with the wind	1939	231	drama
102	Star Wars	1977	124	sci-fi
103	Wayne's World	1992	95	comedy

THE RELATIONAL MODEL

Sample Student Relation

Student(stu_num: integer, stu_lname: string, stu_fname: string, stu_init: char, stu_dob: date, stu_class: string, stu_gpa: real, stu_transfer: boolean, dept_code: string, stu_phone: string, prof_num: integer)

Instance of
Student relation

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1985	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1991	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1979	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1986	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1968	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1989	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1983	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1996	15	Fr	2.92	No	ACCT	2315	230

Attribute Name

STU_NUM = Student number
 STU_LNAME = Student last name
 STU_FNAME = Student first name
 STU_INIT = Student middle initial
 STU_DOB = Student date of birth
 STU_HRS = Credit hours earned
 STU_CLASS = Student classification
 STU_GPA = Grade point average
 STU_TRANSFER = Student transferred from another institution
 DEPT_CODE = Department code
 STU_PHONE = 4-digit campus phone extension
 PROF_NUM = Number of the professor who is the student's advisor

THE RELATIONAL MODEL

Sample Movie Database Schema

Movies. This relation is an extension of the example Movies relation. We have added two new attributes; `studioName` tells us the studio that owns the movie, and `producerC#`.

```
Movies( title:string, year:integer, length:integer, genre:string, studioName:string,  
        producerC#:integer )
```

MovieStar. This relation tells us something about stars. The key is name, the name of the movie star.

```
MovieStar( name:string, address:string, gender:char, birthdate:date )
```

StarsIn. This relation connects movies to the stars of that movie, and likewise connects a star to the movies in which they appeared. Notice that all three attributes are needed to form a key.

```
StarsIn( movieTitle:string, movieYear:integer, starName:string )
```

MovieExec. This relation tells us about movie executives. It contains their name, address, and net worth as data about the executive.

```
MovieExec( name:string, address:string, cert#:integer, netWorth:integer )
```

Studio. This relation tells about movie studios. We rely on no two studios having the same name, and therefore use name as the key. The other attribute is the certificate number for the president of the studio.

```
Studio( name:string, address:string, presC#:integer )
```


DEFINING A RELATIONAL SCHEMA IN SQL

SQL is the principal language used to describe and manipulate relational databases. There is a current standard for SQL is ISO/IEC 9075:2016 (<https://webstore.ansi.org/Standards/ISO/ISOIEC90752016>). Most commercial database management systems implement something similar, but not identical to, the standard. There are two aspects to SQL:

1. The [Data-Definition Language](#) (DDL) for declaring database schemas.
2. The [Data-Manipulation Language](#) (DML) for querying databases and for modifying the database.

DEFINING A RELATIONAL SCHEMA IN SQL

SQL makes a distinction between three kinds of relations:

Stored relations, which are called **tables**. A table is a relation that exists in the database and that can be modified by changing its tuples, as well as queried.

Views, which are relations defined by a computation. These relations are not stored, but are constructed, in whole or in part, when needed.

Temporary tables, which are constructed by the SQL language processor when it performs its job of executing queries and data modifications. These relations are then thrown away and not stored.

DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

ORACLE

Oracle Express Edition Download:

<https://www.oracle.com/database/technologies/xe-downloads.html>

SQL Developer Download:

<https://www.oracle.com/tools/downloads/sqldev-downloads.html>

The Oracle logo, consisting of the word "ORACLE" in a white, sans-serif font with a registered trademark symbol, set against a solid red rectangular background.

ORACLE®

DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

MySQL

MySQL Community Edition Download:

<https://dev.mysql.com/downloads/mysql>

MySQL Workbench Download (developer GUI tool):

<https://dev.mysql.com/downloads/workbench>



DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

SQL Server

SQL Server Express Edition Download:

<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

SQL Server Management Studio Download:

<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>



DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

Microsoft Access

Microsoft Access Trial Version Download:

<https://www.microsoft.com/en-us/microsoft-365/access>



DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

PostgreSQL

PostgreSQL Download:

<https://www.postgresql.org/download>

PGAdmin Download:

<https://www.pgadmin.org/download>



DEFINING A RELATIONAL SCHEMA IN SQL

For this class we will focus on Oracle. Most of the concepts and syntax presented works on most of the modern DBMS. Here are a few examples free to use (except Microsoft Access that requires Office license).

IBM DB2

IBM DB2 Database Download:

<https://www.ibm.com/analytics/db2/trials>

Toad for IBM DB2 Trial Download:

<https://www.quest.com/products/toad-for-ibm-db2>



DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

All attributes in a relation must have a data type.

1. **Character strings** of fixed or varying length. The type `CHAR(n)` denotes a fixed-length string of up to n characters. `VARCHAR(n)` also denotes a string of up to n characters. The difference is implementation-dependent; typically CHAR implies that short strings are padded to make n characters, while VARCHAR implies that an end marker or string-length is used.

`VARCHAR(5) 'abc' => CHAR(5) 'abc '`

DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

2. **Bit strings** of fixed or varying length. These strings are analogous to fixed and varying-length character strings, but their values are strings of bits rather than characters. The type `BIT(n)` (`CHAR(n) FOR BIT DATA` - Oracle) denotes bit strings of length n, while `BIT VARYING(n)` (`VARCHAR(n) FOR BIT DATA` - Oracle) denotes bit strings of length up to n.

DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

3. **BOOLEAN** type denotes an attribute whose value is logical. The possible values of such an attribute are **TRUE**, **FALSE**, and **UNKNOWN**. (Not part of Oracle SQL)

DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

- 4. `INT` or `INTEGER` denotes typical integer values. The type `SHORTINT` also denotes integers, but the number of bits permitted may be less, depending on the implementation

DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

5. Floating-point numbers can be represented in a variety of ways. We may use the type `FLOAT` or `REAL` (these are synonyms) for typical floating point numbers. A higher precision can be obtained with the type `DOUBLE PRECISION`. SQL also has types that are real numbers with a fixed decimal point. For example, `DECIMAL(n,d)` (or `NUMERIC`) allows values that consist of n decimal digits, with the decimal point assumed to be d positions from the right.

123.45 => `DECIMAL(6,2)`

DEFINING A RELATIONAL SCHEMA IN SQL

Data Types

6. Dates and times can be represented by the data types `DATE` and `TIME`, respectively. These values are essentially character strings of a special form. Different SQL implementations may provide many different representations for dates and times, but the following is the SQL standard representation. A date value is the keyword `DATE` followed by a quoted string of a special form.

```
DATE '1948-05-14'    TIME '15:00:02.5'
```

Some DBMS specifics to set the date to 1st Jan 2021

ORACLE: `TO_DATE('01/01/2021','MM/DD/YYYY')`

MS Access: `#01/01/2021#`

Sybase/SQL Server: `'01/01/2021'`

MySQL: `01-01-2021`

IBM DB2: `DATE ('01/01/2021')`

DEFINING A RELATIONAL SCHEMA IN SQL

Table Declaration

The simplest form of declaration of a relation schema consists of the keywords CREATE TABLE followed by the name of the relation and a parenthesized, comma-separated list of the attribute names and their types.

For example, movies relation:

```
Movies (title:string, year:integer, length:integer, genre:string, studioName:string, producerC#:integer )
```

In SQL will be declared as:

```
CREATE TABLE Movies (  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Modifying Relation Schemas

We can delete a relation R by the SQL statement:

```
DROP TABLE R;
```

More frequently than we would drop a relation that is part of a long-lived database, we may need to modify the schema of an existing relation. These modifications are done by a statement that begins with the keywords **ALTER TABLE** and the name of the relation. We then have several options, the most important of which are:

1. **ADD** followed by an attribute name and its data type.
2. **DROP** followed by an attribute name.

DEFINING A RELATIONAL SCHEMA IN SQL

Modifying Relation Schemas

For example, adding the attribute phone to the MovieStar relation:

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

We may add multiple attributes part of the same statement:

```
ALTER TABLE MovieStar ADD phone CHAR(16), eyeColor VARCHAR(30);
```

We may allow or not allow the special NULL value for a relation attribute with:

```
ALTER TABLE MovieStar ADD phone CHAR(16) NULL;
```

If NULL value is not specified the value for the attribute is considered NOT NULL.

DEFINING A RELATIONAL SCHEMA IN SQL

Modifying Relation Schemas

To delete an attribute from a relation you may have to run:

```
ALTER TABLE MovieStar DROP phone;
```

DEFINING A RELATIONAL SCHEMA IN SQL

Default Values

When we create or modify tuples, we sometimes do not have values for all components. There are times when we would prefer to use another choice of default value, the value that appears in a column if no other value is known. In general, any place we declare an attribute and its data type, we may add the keyword **DEFAULT** and an appropriate value. That value is either **NULL** or a constant. Certain other values that are provided by the system, such as the current time, may also be options.

Default values can be declared either part of the table creation or later by modifying the table:

```
CREATE TABLE MovieStar (  
...  
    eyeColor VARCHAR(30) DEFAULT 'brown',  
...  
);
```

```
ALTER TABLE MovieStar ADD eyeColor VARCHAR(30) DEFAULT 'brown';
```

DEFINING A RELATIONAL SCHEMA IN SQL

Declaring Keys

There are two ways to declare an attribute or set of attributes to be a key in the `CREATE TABLE` statement that defines a stored relation.

1. We may declare one attribute to be a key when that attribute is listed in the relation schema.
2. We may add to the list of items declared in the schema (which so far have only been attributes) an additional declaration that says a particular attribute or set of attributes forms the key.

If the key consists of more than one attribute, we have to use method (2). If the key is a single attribute, either method may be used. There are two declarations that may be used to indicate keyness:

a) `PRIMARY KEY`

b) `UNIQUE`

DEFINING A RELATIONAL SCHEMA IN SQL

Declaring Keys

The effect of declaring a set of attributes S to be a key for relation R either using `PRIMARY KEY` or `UNIQUE` is the following:

Two tuples in R cannot agree on all of the attributes in set S, unless one of them is `NULL`. Any attempt to insert or update a tuple that violates this rule causes the DBMS to reject the action that caused the violation.

If `PRIMARY KEY` is used, then attributes in S are not allowed to have `NULL` as a value for their components.

Sample of single-attribute primary key.

```
CREATE TABLE MovieStar (  
    name CHAR(30) PRIMARY KEY,  
    address VARCHAR(255),  
    gender CHAR(1),  
    birthdate DATE  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Declaring Keys

Example of declaring a composite key part of Movies relation:

```
CREATE TABLE Movies (  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT,  
    CONSTRAINT PK_title_movies PRIMARY KEY (title, year)  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Declaring Keys

Example of declaring a composite UNIQUE key on Movies table:

```
CREATE TABLE Movies (  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT,  
    CONSTRAINT UK_Movies UNIQUE (title, length, genre)  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Add Keys to existing tables

```
ALTER TABLE Movies ADD  
    CONSTRAINT PK_title_movies PRIMARY KEY (title, year);
```

```
ALTER TABLE Movies ADD  
    CONSTRAINT UK_Movies UNIQUE (title, length, genre);
```


DEFINING A RELATIONAL SCHEMA IN SQL

Drop Keys from existing tables

```
ALTER TABLE Movies DROP CONSTRAINT PK_title_movies;
```

```
ALTER TABLE Movies DROP CONSTRAINT UK_Movies;
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Constraints helps to restrict the data that may be stored in a database. So far, we have seen only one kind of constraint, the requirement that an attribute or attributes form a key.

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Foreign Key Constraints

A common kind of constraint, called foreign key constraint, asserts that a value appearing in one context also appears in another, related context. For example, in our movies database, should we see a `StarsIn` tuple that has person `p` in the `starName` component, we would expect that `p` appears as the name of some star in the `MovieStar` relation. If not, then we would question whether the listed "star" really was a star.

```
StarsIn( movieTitle:string, movieYear:integer, starName:string )
```

```
MovieStar( name:string, address:string, gender:char, birthdate:date )
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Foreign Key Constraints

```
StarsIn( movieTitle:string, movieYear:integer, starName:string )
```

```
MovieStar( name:string, address:string, gender:char, birthdate:date )
```

```
CREATE TABLE StarsIn (  
    movieTitle VARCHAR(100),  
    movieYear INT,  
    starName VARCHAR(100),  
    PRIMARY KEY (movieTitle, movieYear),  
    CONSTRAINT FK_StarName FOREIGN KEY (starName)  
    REFERENCES MovieStar (name)  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Foreign Key Constraints

```
StarsIn( movieTitle:string, movieYear:integer, starName:string )
```

```
MovieStar( name:string, address:string, gender:char, birthdate:date )
```

SQL Modify Table Add Foreign Key

```
ALTER TABLE StarsIn ADD  
    CONSTRAINT FK_StarName  
    FOREIGN KEY (starName) REFERENCES MovieStar (name);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Foreign Key Constraints

```
StarsIn( movieTitle:string, movieYear:integer, starName:string )
```

```
MovieStar( name:string, address:string, gender:char, birthdate:date )
```

SQL Modify Table DROP Foreign Key

SQL Server/Oracle/Sybase/MS Access/PostgreSQL

```
ALTER TABLE StarsIn DROP  
CONSTRAINT FK_StarName;
```

MySQL/IBM DB2

```
ALTER TABLE StarsIn DROP  
FOREIGN KEY FK_StarName;
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

NULL / NOT NULL Constraints

Allows (or not) null value for a given attribute.

SQL NULL/NOT NULL Create Table.

```
CREATE TABLE StarsIn (  
    ...  
    starName VARCHAR(100) NOT NULL,  
    roleName VARCHAR(100) NULL,  
    ...  
)
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

NULL / NOT NULL Constraints

Allows (or not) null value for a given attribute.

SQL NULL/NOT NULL Alter Table.

```
ALTER TABLE StarsIn ADD starName VARCHAR(100) NOT NULL;
```

```
ALTER TABLE StarsIn ADD roleName VARCHAR(100) NULL;
```


DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

NULL / NOT NULL Constraints

Allows (or not) null value for a given attribute.

SQL NULL/NOT NULL Drop Constraints.

```
ALTER TABLE StarsIn MODIFY starName VARCHAR(100) NULL;
```

```
ALTER TABLE StarsIn MODIFY roleName VARCHAR(100) NOT NULL;
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

CHECK Constraints

Limits value for a given attribute.

SQL CHECK Constraints CREATE TABLE

```
CREATE TABLE StarsIn (  
    movieTitle VARCHAR(100),  
    movieYear INT,  
    starName VARCHAR(100),  
    PRIMARY KEY (movieTitle, movieYear),  
    CONSTRAINT FK_StarName FOREIGN KEY (starName)  
        REFERENCES MovieStar (name),  
    CONSTRAINT CK_StarIn_Year  
        CHECK (movieYear>=1900 and movieYear<2500)  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

CHECK Constraints

Limits value for a given attribute.

SQL CHECK Constraints ALTER TABLE

```
ALTER TABLE StarsIn ADD CONSTRAINT CK_StarIn_Year  
                CHECK (movieYear>=1900 and movieYear<2500) ;
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

CHECK Constraints

Limits value for a given attribute.

SQL CHECK Constraints DROP Constraint

```
ALTER TABLE StarsIn DROP CONSTRAINT CK_StarIn_Year
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

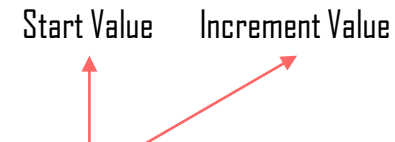
Auto-Increment

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

SQL Auto Increment – create table

Start Value Increment Value

```
CREATE TABLE Movies (  
    movieID int IDENTITY(1,1) PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```



SQL Server/Sybase

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment

SQL Auto Increment – create table

```
CREATE TABLE Movies (  
    movieID int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

Start Value



```
ALTER TABLE Movies AUTO_INCREMENT=20;
```

MySQL

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment

SQL Auto Increment – create table

```
CREATE TABLE Movies (  
    movieID int NOT NULL AUTO_INCREMENT (1,1) PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

MS Access

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment


SQL Auto Increment – create table

```
CREATE SEQUENCE seq_movies  
    MINVALUE 1  
    START WITH 1  
    INCREMENT BY 1  
    CACHE 10;
```

Oracle

Getting the next sequence number, used for example part of the SELECT statement fields.

seq_movies.**nextval**



DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment

SQL Auto Increment – create table

IBM DB2

```
CREATE TABLE Movies (  
    movieID integer NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1)  
    PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment

SQL Auto Increment – create table

```
CREATE TABLE Movies (  
    movieID SERIAL NOT NULL PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

PostgreSQL

Serial increase the value even in case of failure resulting in sequence fragmentation.

DEFINING A RELATIONAL SCHEMA IN SQL

Constraints on Relations

Auto-Increment

SQL Auto Increment – create table

```
CREATE SEQUENCE seq_movies;
```

```
CREATE TABLE Movies (  
    movieID NOT NULL  
        DEFAULT nextval('seq_movies') PRIMARY KEY,  
    title CHAR(100),  
    year INT,  
    length INT,  
    genre CHAR(10),  
    studioName CHAR(30),  
    producerC# INT  
);
```

PostgreSQL