The Relational Data Model and Relational Database Constraints

First introduced by Ted Codd from IBM Research in 1970, seminal paper, which introduced the Relational Model of Data representation. It is based on Predicate Logic and set theory. Predicate logic is used extensively in mathematics and provides a framework in which an assertion can be verified as either true or false. For example, a person with T number T00178477 is named Lila Ghemri. This assertion can easily be demonstrated as true or false.

Set theory deals with sets and is used as a basis for data manipulation in the relational model. For example, assume that A= \{16, 24, 77\}, B= \{44, 77, 90, 11\}. Given this information, we can compute that A \cap B= \{77\}

Based on these concepts, the relational model has three well defined components:

1. A logical data structure represented by relations
2. A set of integrity rules to enforce that the data are and remain consistent over time
3. A set of operations that defines how data are manipulated.

Relational Model Concepts:

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or a flat file of records.

When a relation is thought of as a table of values, each row in the table represents a collection of related data values corresponding to a real world entity. The table name and column names are used to help interpret the meaning of the values in each row.

For example the table below is called STUDENT because each row represents facts about a particular student entity.
The column names –Name, Student_number, Class and Major specify how to interpret the data values in each row based on the column in which they are. All values in a column are of the same data type.

Terminology:

In a formal relational model, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the type of values that can appear in each column is represented by a domain of possible values.

Domains, Attributes, Tuples and Relations:

A domain D is a set of atomic values. An atomic value is a value that is indivisible as far as the formal definition is concerned.

Domains:

A common method of specifying a domain is to define the datatype from which the data values are drawn. It is useful to specify a name for the domain:

-Usa_phone_number: the set of 10 digits numbers valid in the US.
-Local_phone_number: the set of 7 digits phone numbers valid with a particular area code. ddd-dddd
-Social-Security-numbers: the set of valid 9 digits SSN : ddd-dd-dddd
-Names: The set of characters or a string
-Grade_point_average: a float between 0 and 4
-Employee_age: an integer between 16-70

These names are called logical definitions of domains.

A data type or format is also specified for each domain. For example, the data type for the domain USA-Phone_numbers can be declared as a character string of the form (ddd)ddd-dddd, where each d is a digit, and the first three digits form a valid area code.
A domain is thus defined by a name, a data type and a format.

A relation schema or relation scheme $R$, denoted $R(A_1, A_2, \ldots, A_n)$, is made up of a relation name $R$ and a list of attributes $A_1, A_2, \ldots, A_n$.

Each attribute $A_i$ is the name of a feature describing the concept or relation. $D$ is the domain in which each attribute takes its values and is denoted by $\text{dom}(A_i)$.

A relation schema is used to describe a relation: $R$ is called the name of this relation. The degree (or arity) of the relation is the number of attributes $n$ of its relation schema.

A relation of degree seven, which stores information about university students would contain seven attributes describing each student, as follows:

STUDENT(Name, Ssn, Home_phone, Address, Office_Phone, Age, Gpa)

Adding the data type of each attribute, the definition is sometimes written as:

STUDENT(Name:string, Ssn:string, Home_phone:string, Address:string, Office_Phone:string, Age:integer, Gpa:real)

In this relation schema, STUDENT is the name of the relation, which has 7 attributes. We have assigned generic data types to these attributes. However, we can also use the previously defined domains for some of these attributes. For example: $\text{dom}(\text{Name}) = \text{Names}; \text{dom}(\text{Ssn}) = \text{Social_security_numbers}; \text{dom}(\text{HomePhone}) = \text{USA_phone_numbers}$, etc..

A relation (or relation state) $r$ of the relation schema $R(A_1, A_2, \ldots, A_n)$, also denoted by $r(R)$, is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$. Each n-tuple is an ordered list of n-values $t = \langle v_1, v_2, \ldots, v_n \rangle$, where each value $v_i, 1 \leq i \leq n$ is an element of $\text{dom}(A_i)$ or a special value NULL. The $i^{th}$ value in tuple $t$, which corresponds to the attribute $A_i$, is referred to as $t[A_i]$ or $t[i]$. 
The terms relation intension for schema R or relation extension for a relation state r(R) are also used.

STUDENT

<table>
<thead>
<tr>
<th>Name</th>
<th>Ssn</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benjamine Bayer</td>
<td>304-61-2435</td>
<td>(817)373-1616</td>
<td>2918 Bluebonet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
<tr>
<td>Chung-cha Kim</td>
<td>381-62-1245</td>
<td>(817)375-4408</td>
<td>124 Kirby Road</td>
<td>NULL</td>
<td>18</td>
<td>2.89</td>
</tr>
<tr>
<td>Dick Davidso n</td>
<td>489-11-2320</td>
<td>NULL</td>
<td>3425 Elgin Road</td>
<td>(817) 748-6592</td>
<td>25</td>
<td>3.53</td>
</tr>
<tr>
<td>Rohan Panchal</td>
<td>489-22-1100</td>
<td>(817) 378-9034</td>
<td>238 Lark Avenue</td>
<td>(817)748-6492</td>
<td>28</td>
<td>3.93</td>
</tr>
<tr>
<td>Barbara Benson</td>
<td>533-69-128</td>
<td>(817)856-0101</td>
<td>8568 Lark Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
</tbody>
</table>

STUDENT is the relation name

Name, Ssn, Home_phone, Address, Office_phone, Age and Gpa are attributes

All data entries are tuples.

It is possible for several attributes to have the same domain, the attribute name however will indicate their role or purpose in the relation.

Characteristics of a Relation:

-Ordering of tuples in a relation: A relation is defined as a set of tuples. Mathematically, the elements of a set have no order, hence tuples in a particular relation have no particular order. Tuple ordering is not part of the relation.

Ordering of Values in a tuple and an Alternative definition of a relation:
An n-tuple is an ordered list of n values, so the ordering of values in a tuple is important. The order of values of a tuple can be changed as long as the correspondence between attributes and values is maintained.

An alternative definition of a relation can be given as $R=\{A_1, A_2, \ldots, A_n\}$ a set of attributes (instead of a list) and a relation state $r(R)$ is a finite set of mapping $r=\{t_1, t_2, \ldots, t_m\}$, where each tuple $t_i$ is a mapping from $R$ to $D$ and $D$ is the union of the attribute domains $D=\text{dom}(A_1) \cup \text{dom}(A_2) \cup \ldots \cup \text{dom}(A_n)$

According to the definition of tuple as a mapping, a tuple can be considered as a set of (<attribute> <value>) pairs, where each pair gives the value of the mapping from attribute $A_i$ to a value $v_i$ from $\text{dom}(A_i)$

**Example**

$T=\{(\text{Name, Benjamin Bayer}), (\text{Ssn, 304-61-2435}), (\text{Home_phone, (817)373-1616}), (\text{Address, 2918 Bluebonet Lane}), (\text{Office_phone, NULL}), (\text{Age, 19}), (\text{Gpa, 3.21})\}$

**Find an identical tuple?**

$T=\{(\text{Home_phone, (817)373-1616}), (\text{Name, Benjamin Bayer}), (\text{Age, 19}), (\text{Ssn, 304-61-2435}), (\text{Address, 2918 Bluebonet Lane}), (\text{Office_phone, NULL}), (\text{Gpa, 3.21})\}$

$T=(\text{Benjamin Bayer, 304-61-2435, (817)373-1616, 2918 Bluebonnet Lane, 19, 3.21})$

**Interpretation of a Relation:**

The relation schema can be interpreted as a declaration or an assertion.

Some relations may represent facts about entities, whereas other relations may represent facts about relationships. For example a relation STUDENT represents facts about a student entity. However a relation MAJORS(Student_Ssn, Department_code) asserts that a student majors in
an academic discipline. A tuple in this relation relates a student to the department in which the major is offered.

Relational Model Notation:

- A relation schema $R$ of degree $n$ is denoted by $R(A_1, A_2, \ldots, A_n)$,
- The uppercase letters $Q$, $R$, $S$ denote relation names
- The letters $t$, $u$, $v$ denote tuples
- In general, the name of a relation schema such as $STUDENT$ also includes the current set of tuples in that relation, whereas $STUDENT(\text{Name, Ssn, \ldots})$ refers to the relation schema only.
- An attribute $A$ can be related to the relation name $R$ to which it belongs by using the dot notation $R.A$, example: $STUDENT.\text{Name}$ or $STUDENT.\text{Age}$
- All attribute names in a particular relation must be distinct.
- An $n$-tuple $t$ in a relation $r(R)$ is denoted $t=<v_1, v_2, \ldots, v_n>$ where each value $v_i \ 1 \leq i \leq n$ is the value corresponding to attribute $A_i$. This notation is called component values of tuples.
- Both $t[A]$ or $t. A_i$ (or $t[i]$) refer to the value $v_i$ in $t$ for attribute $A_i$.
- Both $t[A_u, A_v, \ldots, A_z]$ and $t. (A_u, A_v, \ldots, A_z)$, where, $A_u, A_v, \ldots, A_z$ is a list of attributes from $R$, refer to the subtuple of values $<v_u, v_v, \ldots, v_z>$ from $t$ corresponding to the values of the attributes specified in the list.

Relational Model Constraints

In a relational database, there will be many relations and the tuples in these relations are usually related in various ways. Therefore, there must be many restrictions on the actual values in a database, so as to keep the database consistent. These constraints are derived from the application that the database represents.

Constraints on databases can generally be divided into three main categories:

1. Constraints that are inherent in the data model, called inherent-model-based constraints or implicit constraints. For
example, that not 2 tuples should be identical are such constraints.

2. Constraints that are directly expressed in schema of the data model, by specifying them in the DLL (data definition language); these are called schema-based constraints or explicit constraints.

3. Constraints that cannot be directly expressed in the schemas of the data model and must be enforced by the application program, these are called application-based or semantic constraints or business rules.

**Domain Constraints:**
Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains include standard numeric datatypes for integers, real numbers, characters, strings, Booleans. Also are available dates, time, money, a subrange of values from a data type, or an enumerated data type in which all valid values are explicitly listed.

**Key Constraints**
In the formal relational model, a relation is defined as a set of tuples. All elements in a set are, by definition, distinct; hence all tuples in a relation must be distinct. This means that no 2 tuples can have the same combination of values for all their attributes. Usually, there exists a subset of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes. Suppose that we denote SK such a subset; then given any two distinct tuples ti and tj in a relation state r in R we have the constraint that:

\[ ti[SK] \neq tj[SK] \]
Any such set of attributes $SK$ is called a **superkey** of the relation schema $R$.

A superkey $SK$ specifies a uniqueness constraint that not two distinct tuples in any state $r$ of $R$ can have the same value for $SK$.

Every relation has at least one default superkey - the set of all its attributes.

A superkey can have redundant attributes, so a more useful concept is that of a **key**, which has no redundancy.

A **key** $K$ of a relation schema $R$ is a superkey of $R$ with the additional property that removing any attributes $A$ from $K$ leaves a set of attributes $K'$ that is not a superkey of $R$.

A key has to satisfy two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes of the key. This property also applies to a superkey.

2. A key is a minimal superkey - that is a superkey from which we cannot remove any attribute and still have the uniqueness constraint in 1 hold. This property is not required by the superkey.

<table>
<thead>
<tr>
<th>Name</th>
<th>Ssn</th>
<th>Home_phone</th>
<th>Address</th>
<th>Office_phone</th>
<th>Age</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benjamin Bayer</td>
<td>304-61-2435</td>
<td>(817)373-1616</td>
<td>2918 Bluebonet Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.21</td>
</tr>
<tr>
<td>Chung-Cha Kim</td>
<td>381-62-1245</td>
<td>(817)375-4408</td>
<td>124 Kirby Road</td>
<td>NULL</td>
<td>18</td>
<td>2.89</td>
</tr>
<tr>
<td>Dick Davidsson</td>
<td>489-11-2320</td>
<td>NULL</td>
<td>3425 Elgin Road</td>
<td>(817) 748-6592</td>
<td>25</td>
<td>3.53</td>
</tr>
<tr>
<td>Rohan Panchal</td>
<td>489-22-1100</td>
<td>(817)378-9034</td>
<td>238 Lark Lane</td>
<td>(817)748-6492</td>
<td>28</td>
<td>3.93</td>
</tr>
<tr>
<td>Barbara Benson</td>
<td>533-69-128</td>
<td>(817)856-0101</td>
<td>238 Lark Lane</td>
<td>NULL</td>
<td>19</td>
<td>3.25</td>
</tr>
</tbody>
</table>

STUDENT relation: \{Ssn\} is a key of STUDENT. Any set of attributes that includes Ssn – \{Ssn, Name, Age\} is a superkey. \{Address, Name, Age\} is not a key- Why?
A key with multiple attributes must require all its attributes together to have the uniqueness property. Any superkey formed from a single attribute is also a key.

The value of a key attribute is used to identify uniquely each tuple in the relation. A set of attributes constituting a key is a property of the relation, it is a constraint that should hold on every valid relation state of the schema. The key has the property to be **time-invariant**: it must continue to hold when we insert new tuples in the relation.

In general a relation schema may have more than one key, in this case each key is called a **candidate key**. We usually designate one of the candidate keys as the **primary key**. This is the key used to identify tuples in a relation. By convention, the attributes that form the primary key of a relation schema are underlined.

STUDENT { Ssn, Name, Home_Address, Home_Phone, Office_Phone, Age, Gpa}

When a relation schema has several candidate keys, it is better to choose as the primary key the one that has a single attribute, or a small number of attributes. The other candidate keys are designated as unique keys and are not underlined.

**Constraints on NULL Values:**
Another constraint on attributes specifies whether NULL values are or are not permitted. For example, if every student tuple must have a valid, non-NULL value for the Name attribute, then the Name of STUDENT is constrained to be NOT NULL.

**Relational Databases and Relational Database Schemas:**
The definition and constraints we have seen so far apply to single relations and their attributes. A relational database usually contains many relations, with tuples in relations related across relations.
A relational database schema \( S \) is a set of relation schemas \( S= \{ R_1, R_2, \ldots \ R_m \} \) and a set of integrity constraints \( IC \). When we refer to relational database, we implicitly include both its schema and its current state.

A database state that does not obey all the integrity constraints is called an \textbf{invalid state}. A state that satisfies all the constraints in the defined set of integrity constraints \( IC \) is called a \textbf{valid state}.

\textbf{EMPLOYEE (FName,Minit, LName, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)\\}
\textbf{DEPARTEMENT (Dname, Dnumber, Mgr_ssn, Mgr_start_date)}

Often different relations refer to the same real world concept. For example, the number given to a department is called \( Dno \) in \textbf{EMPLOYEE} and \( Dnumber \) in \textbf{DEPARTMENT}. Attributes that represent the same real-world concept may or may not have identical names in different relations.

\textbf{Integrity, Referential Integrity and Foreign Keys:}

The \textbf{entity integrity} constraint states that \textbf{no primary key value can be NULL}. This is because the primary key is used to identify tuples.

Key constraints and entity integrity constraints are specified on individual relations.

\textbf{Referential integrity} constraints are specified between \textbf{two relations} and are used to maintain consistency among tuples in the two relations.

Informally referential integrity constraints state that a tuple in one relation that refers to another relation, must refer to an existing tuple in that relation.

To define referential integrity more formally, we first define the concept of foreign key.
**Foreign Key:**

A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to refer to the relation R2.
2. A value of FK in a tuple \( t_1 \) of the current state \( r_1(R1) \) either occurs as a value of PK for some tuple \( t_2 \) in the current state \( r_2 \) of R2 or is NULL. In the former case, we have \( t_1[FK] = t_2[PK] \), and we say that the tuple \( t_1 \) references or refers to the tuple \( t_2 \).

In this definition, R1 is called the referencing relation and R2 the referenced relation. If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold. In a database with many relations, there are usually many such referential integrity constraints.

To specify these constraints, first we must have a clear understanding of the meaning or role of each attribute or set of attributes plays in the various relation schemas of the database.

Referential integrity constraints typically arise from the relationships among the entities represented by the relation schemas.

**EMPLOYEE (FName, Minit, LName, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)**

**DEPARTMENT (Dname, Dnumber, Mgr_Ssn, Mgr_start_date)**

**PROJECT (Pname, Pnumber, Plocation, Dnum)**

**DEPT_LOCATIONS (Dnumber, Dlocation)**

**WORKS_ON (Essn, Pno, Hours)**

**DEPENDENT(Essn, Dependent_name, Sex, Bdate, Relationship)**

For example, in the EMPLOYEE relation, the attribute Dno refers to the department in which the employee works; hence we designate Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT relation. This
means that a value of Dno in any tuple \( t_i \) of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT- the Dnumber – attribute in some tuple \( t_j \) of the DEPARTMENT relation, or the value Dno can be NULL if the employee does not belong to any department or will be assigned a department later.

A foreign key can refer to its own relation. For example, the attribute Super_Ssn in Employee refers to the supervisor of an employee. That is another employee, represented by a tuple in the relation EMPLOYEE. Hence Super_ssn is a foreign key that reference the EMPLOYEE relation itself.

Diagram display of referential integrity constraint by drawing a directed edge from the foreign key to the relation it references.

Surrogate Keys:

Surrogate keys are keys that are “created”. They are numeric values that are appended to the relation to serve as the primary key. Surrogate keys have no meaning to the user and are normally hidden from them on forms or queries and reports. They are widely used in order to avoid the overhead in memory of using composite keys.

Most DBMS systems have a facility for automatically generating surrogate key values.

All integrity constrains should be specified on the relational database schemas, as part of the database definition.

Most relational databases support key, entity integrity and referential integrity constraints.

**Other types of Constraints:**

Semantic integrity constraints are constraints that are application specific and can be enforced within the application programs that update the
database. An example of such constraints is that the maximum number of hours in that an employee can work in all projects is 56.

Another type of constraints is the functional dependency constraint, which establishes a functional relationship among two sets of attributes X and Y. This constraint specifies that the value of X determines a unique value of Y in all states of the relation. It is denoted as a functional dependency \( X \rightarrow Y \).

**Database Modification and Update:**

There are 3 operations that can change the state of relations in the database: **Insert**: to add new data into the relation by adding one or more tuples in a relation.

**Delete**: is used to delete tuples

**Update (or Modify)**: used to change the value of some attributes in existing tuples.

Whenever these operations are applied, the integrity constraints defined on the relational database should not be violated.

**The Insert Operation:**

The Insert operation provides a list of attribute values for a tuple t that is to be inserted into a relation R. An Insert operation can violate any of the specified integrity constraints described earlier:

Example:

Insert <' Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6578 Windy Lane, Katy, TX', 'F', 28000, NULL, 4> into EMPLOYEE

Result: Rejected why?

Insert <' Alicia', 'J', 'Zelaya', 999887777, '1960-04-05', '6578 Windy Lane, Katy, TX', 'F', 28000, NULL, 4> into EMPLOYEE

Result: Rejected Why?

Insert <' Cecilia', 'F', 'Kolonsky', 677898790, '1960-04-05', '6578 Windy Lane, Katy, TX', 'F', 28000, 987654321, 7> into EMPLOYEE
Result: Reject why?


Result: OK

If an insertion violates one or more constraints, the default option is to reject the insertion.

The Delete Operation:
The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion a condition on the attribute of the relation selects the tuple or tuples to be deleted.

- Operation:
  Delete the WORKS-ON tuple with Essn= ‘999887777’ and Pno=10.
  Result: ?? works
  DELETE from WORKS-ON where Essn= ‘999887777’ AND Pno= 10;
  doesn’t work
- Operation:
  Delete the EMPLOYEE tuple with Ssn= ‘999887777’
  Result: ??
- Operation:
  Delete the Employee tuple with Ssn= ‘333445555’
  Result: ??

Several options are available if a deletion operation causes a violation. The first option, called restrict, is to reject the deletion. The second option, called cascade, is to attempt to propagate the deletion by deleting tuples that reference the tuple being deleted.

A third option, called set null or set default is to modify the referencing attribute values that cause the violation; each such value is set to NULL or changed to another default valid tuple.
Combinations of these three options are also possible, in general the DBMS will allow the database designer to specify which options applies in case of violation of the constraints.

**The Update Operation**
The Update or (Modify) operation is used to change the values of one or more attributes in a tuple or tuples for some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple or tuples to be modified.

- Operation: Update the Salary of the EMPLOYEE tuple with Ssn=‘999887777’ to 28000
  
  Result: ?

- Operation: Update the Dno of the EMPLOYEE tuple with Ssn=‘999887777’ to 1
  
  Result: ?

- Operation: Update the Dno of the EMPLOYEE tuple with Ssn=‘999887777’ to 7
  
  Result: ?

- Operation: Update the Ssn of the EMPLOYEE tuple with Ssn=‘999887777’ to ‘987654321’
  
  Result: ?

Updating an attribute that is neither part of a primary key or a foreign key usually causes no problems; the DBMS need to check and confirm that the new value is of the correct data type and domain. Modifying a primary key is similar to deleting one tuple and inserting another in its place because we use the primary key to identify the tuples. If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation (or set to NULL).