02. Relational Model

[PM Jat, DAIICT, Gandhinagar]

Content:

- What is a Relation
- Relation Characteristics
- Relation instance and relation schema
- Relation constraints
- Schema Examples
- Languages for performing operations on relations

What is a relation?

The origin of the relational model lies in “relation” in mathematics. The model was first proposed by Dr. E.F. Codd of IBM Research through the paper "A Relational Model for Large Shared Data Banks" in Communications of the ACM, June 1970. System R was the first

To understand what the *relation* in the relational model is, let us begin with the relation in math.

- Consider four sets A, B, C, and D;
- Relation is defined as (a, b, c, d) where
  a ∈ A, b ∈ B, c ∈ C, and d ∈ D.
- Or, relation is a subset of (A x B x C x D).

Now let us try using this concept in some real situations. Assume that

- Set A is a universal set of student IDs.
- Set B is a universal set of student names
- C as a universal set of Program Codes that the university offers, and
- D is a set of real values (0 to 10).

With this premise shown below is a relation drawn from said four sets

```
(101,Rahul,BCS,7.5)
(102,Vikash,BIT,8.6)
(103,Shally,BEE,5.4)
(104,Alka,BIT,6.8)
(105,Ravi,BCS,6.5)
```

Each element in this relation is an “n-tuple”; i.e., ordered list values drawn from respective sets. “n-tuple” may simply be called a “tuple”.

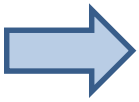
A tuple represents some facts that are to be recorded in the database. In the relation above, each tuple represents a student entity.

"Relation" in the relational model is an adaption of "relation" in math. Here are some of the adaptions/characterizations that are done in the relation model:

1. We say that each tuple expresses/represents some fact about "something" and captures a database fact. In the example here, the tuple represents a student entity.
2. We bring in a notion of "attribute"; we say that each value in the tuple is a value for the corresponding attribute of the entity or event (next example). In the example here, the first is the value for the ID attribute, second is value for name attribute, and so forth.
3. Let us call corresponding sets of value in a relation as "domain" for the attribute
4. We give a name to a relation based on what it represents; in this case, we name this relation as "Student".

The figure below depicts "tuple-view" vis-à-vis "tabular-view" of Student relation. Relation can be viewed as a table, that is why probably, relation is also called as *table*. A tuple is also called as row in a table. Attribute is called as column.

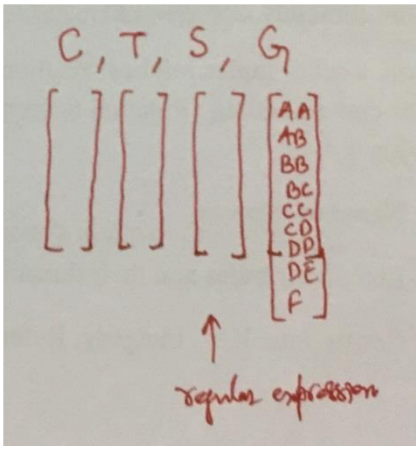
(101, Rahul, BCS, 7.5)
(102, Vikash, BIT, 8.6)
(103, Shally, BEE, 5.4)
(104, Alka, BIT, 6.8)
(105, Ravi, BCS, 6.5)



ID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5

Terminology jargon: While terms relation, attribute, tuple is more used in theoretical discussions. table, column, and row are used in Structured Query Language (SQL). We often, in tabular view, show header information of table, i.e., attribute or column names. We may also give domain for each attribute. Note that SQL is a query language that is used for "data definition" and "data manipulation" in relational model.

Another example:



Consider four sets C, T, S, G; where

C is set of course numbers (offered in an institute)

T is set of term-ids

S set of student IDs

G is set of grades

Following is a relation drawn from these sets -

(IT214, Autumn-2017, 201501123, BB)

(IT214, Autumn-2016, 201501123, CD)

…

Each tuple, in this case, represents a fact of an event of "student registering in a course offered in a term". We name this relation as **Registrations**, i.e. registrations of students in a course

Below is depiction of some snapshot of relation instance.

Coursera	Term	StudentID	Grade
IT214	Autumn-2017	201501123	DE
IT214	Autumn-2016	201501123	BC
…			

Relation Characteristics

A **relation** is a set of tuples.

The order of tuples is not important in a relation.

A tuple is an ordered list of values (for respective attribute) drawn from the corresponding domain for the attribute.

A tuple can also be represented as a set of (Attribute-Name, attribute-value pairs).
Example: {(ID,101), (ProgID, BCS), (Name, Rahul), (CPI,7.5)}

All tuples are distinct from one another "in terms of values". This means no two tuples can be identical.

Relation instance and relation schema

In the relational model, the relation, a set of tuples, as understood above, implies "**relation instance**". A relations instance holds data.

Every relation also has a corresponding "**Relation Schema**". The relation schema describes the following –

- Name of relation
- List of attributes and their domain, and
- Constraints: Key, Integrity, Referential Integrity, and so

Example:

> Student (Studentid [int], name [varchar(20)], progid [smallint], batch [smallint], cpi, [numeric(5,2]) with other details.
>
> Note: in SQL, domains are defining through SQL data types.

Notations:

> Schema: `R(A1, A2,... An)`, where `R` is name of relation and `A1` through `An` are name of attribute in relation `R`
>
> Domain for attribute `Ai` is defined as `dom(Ai)`
>
> `r(R)` or only `r` is used for relation [instance] of schema R

Tuple: two ways to represent a tuple

As ordered list of n-values; `t = <v1, v2, ..., vn>` ; or

As a set of n (attribute-name, attribute-value) pairs:
`t = {(A2,v2), (A1,v1), ..., (An,vn)}`

Notationally, the value of attribute `Ai` in tuple `t`, can be expressed as `t[Ai]` or `t.Ai`

Similarly, `t[Au, Av, ..., Aw]` refers to the sub-tuple of t containing the values of attributes `Au, Av, ..., Aw`, respectively in `t`

Relation constraints

Recall

- "Constraints are data existential rules" and are also called "Database Integrity Constraints".
- Constraints are specified as part of the schema, and DBMS is to ensure compliance with constraints on every valid state of the database.

Following are types of constraints specified on a relation-

1. Domain constraints
2. Key constraints and Entity Integrity constraints
3. **NOT NULL** value constraints
4. Referential Integrity Constraints
5. Other constraints.

Terminology note: the term "relation" implies "relation instance"; for relation schema, we explicitly augment "relation" with term "schema". However sometime, people use term "relation" while they mean "relation schema". In most cases by context, we should be able detect if relation refers to "relation schema".

Domain constraints

- Attribute values are drawn from their corresponding domain.

NOT NULL constraints

- NULL is a special value in databases. NULL simply, attributes for which we do not have value, contains NULL value.
- NOT NULL constraints implies that the attribute cannot have NULL value. That is, value for such attribute mandatory in a tuple. A tuple cannot exist in the database without having value for such attributes.

Key Constraints

Primarily this constraint requires that every tuple in a relation is distinct.

DBMS needs to ensure that no two tuples are same. This is either done by comparing all attribute values or by comparing fewer attribute values. To find out correct set of fewer values, we introduce two terms

- Candidate Key or simply "Key"
- Super Key

Super Key is some group (subset) of attribute from attributes of a relation R, such that no two tuples can have same value for that set of attributes. Fewer the attribute, better it is, because the checking for uniqueness would be more efficient. Is not it? Obviously, Yes!

Formally we can express super key as following:

Suppose t1 and t2 are two tuples from relation instance r of schema R. Assume R represents here all set of attributes of Relation Schema R.

Let SK ⊆ R

Then property of SK is that t1[SK] != t2[SK]

Make sure that you understand the expression "`t1[SK] != t2[SK]`"

As an example, consider our relations `STUDENT(SID, Name, DOB, CPI)`

The following are super key. Note meaning of the attribute determines whether it is correct or not?

```
{SID, Name, DOB, CPI} is super key
{SID, Name, CPI} is super key
{SID, Name, DOB} is super Key
{SID, Name} is super key
{SID} is super key [Minimal Super Key]
{Name, DOB, CPI} is Not super key
{DOB, CPI} is NOT super key
```

You must have observed here is that there are too many possible "attributes combination" that are super-key. However, one this is common in all that is SID.

Now this dilemma of so many possible super keys; we define another type of Key, called Candidate Key.

Candidate Key is a Super Key such that there is no "redundant" attributes. Redundancy here means if we remove an attribute from super key and then remaining attributes are still super key then we say that attribute is redundant.

So, here we further say that Candidate Key is Super Key with minimum set of attributes.

Let us also say it as following: value for a candidate key (CK) attribute is unique in a relation instance! Make senses?

Candidate Key is a Super Key also; only the thing is it is minimal.

Mathematically, SK is super set of CK is always true!

SK ⊇ CK

Note that candidate key has more practical usage than super key; super is a theoretical concept and more often used for various theoretical explanations.

Remember: when I say "Key", it means "candidate key", not super key!

Let us consider an example where candidate key is composite. Consider a situation of recording sales details. One of the relations in such a situation will be a relation "invoice-details" and would typically have following set of attributes. Figure also shows some values – should help you in understand the thing better. Can you guess a candidate key here?

InvNo	ICODE	Qty	Price
1	i001	20	10000
1	i002	30	6000
1	i003	100	25000
2	i002	50	10000
3	i003	100	25000
3	i004	40	8000
4	i005	15	82500

In this case candidate key shall be: `{InvNo,ICODE}`; includes two attributes.

This is drawn from the following facts from problem domain-

- An invoice may have multiple items; therefore, invoice number can repeat in this relation.
- An item can appear in multiple invoices; therefore, item-code can repeat in the relation.
- BUT together they shall never repeat.

It may be worth noting here is that there is an assumption; that an item shall not appear twice in an invoice – should be a fair assumption.

Keys having more than one attributes are called **composite keys**.

When key is composite, values for composite attributes individually repeat in a relation instance but they together do not repeat; they are unique in combined manner. In the example of our invoice-details, invoice-number and item-no together do not repeat.

A relation may have multiple candidate keys

As an example, suppose we also record aadhar number (UUID) of students in our student relation, and our relation becomes as following:

```
Student(SID, UUID, Name, DOB, CPI)
```

SID	UUID	Name	ProgID	CPI
101	5433	Rahul	BCS	7.5
102	4312	Vikash	BIT	8.6
103	5218	Shally	BEE	5.4
104	5701	Alka	BIT	6.8
105	5809	Ravi	BCS	6.5

Here we have two candidate keys: CK1 = {SID} and CK2 = {UUID}.

In this no two tuple can have same values for SID and UUID individually!

Here it is very important to note here is that having multiple keys is different than key having composite Key.

Therefore, above is not equal to CK = {SID, UUID}

Exercise:

What is the key of following relation schema?

CourseOfferings(TermID, CourseNo,InstrutorID)
Registrations1(CourseNo, StudentNo, Student Name, Grade, FacultyID)
Registrations2(StudentID, TermID, CourseNo, Grade)

Entity Integrity Constraint:

A purpose of Candidate Key is to "uniquely identifiability" of a tuple in its tuple-set, that is relation.

Entity Integrity Constraint simply tells that Candidate key is also constrained to NOT NULL while other one is "unique". If we allow NULL values is no more possible identifying a tuple in its set (relation). So, this constraint requires that entity is always identifiable.

Primary Key:

In relational model, as such there is no discussion of primary key. It only describes Super Key and Candidate Key. However, when it comes to implementation of databases on some RDBMS, primary key comes into picture.

Candidate key becomes Primary Key.

Also note that NOT NULL constraint is also applied to Primary Key, a tuple cannot value null values for primary key attributes.

When primary key is composite, none of the attribute that is part of primary key can be null.

Referential Integrity Constraint

Foreign Key

A tuple is typically used for representing an entity for example student. Refer diagram below

102	Vikash	BIT	8.6

Set of such tuples represents all students; called tuple-set. Tuple-set represents an entity set.

Entities are often related, and this fact needs to be captured in databases. For example, Student and Program; the relationship between these entities is a student entity gets admission in a program, and studies in that program. The third value in tuple above, which is progid, is basically a reference to program entity, shown below

BIT	BTech(IT)	30	CS

All tuples-sets in relations Student and Program are put together, shown below, is how it should look like. Value in progid attribute basically stores a reference to a program tuple.

Student

StudentID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5

Program

PID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

Let us call Student as referencing relation and Program as referred relation. A tuple in referencing relation stores reference of a tuple in referencing entity.

For storing reference, we have candidate key of referenced relation as **foreign key** in referencing relation.

Revise: What is a Foreign Key?

Referential Integrity Constraint

Referential Integrity constraint requires that a value in a foreign key in a tuple should refer to some existing tuple; a tuple that is identified by value in foreign key. Figure below shows a student tuple that has "invalid" value in foreign key. Invalid because it attempts to refer a tuple that does not exist. A new a tuple, show below, having progid=BEC, cannot be added to student relation as FK here (progid) has a value that does not refer to a existing tuple in Program relation.

Student

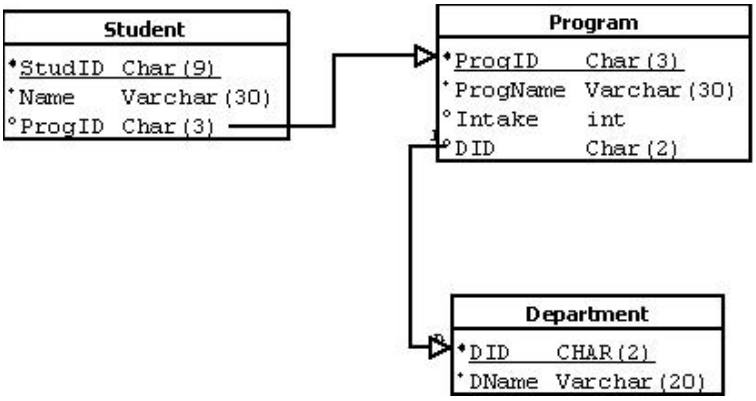
StudentID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5
109	Punit	BEC	7.8

Program

PID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

Formally we can explain referential integrity as following: Suppose relation R has a foreign key FK, that refers to attribute K in relation S. Suppose t1 is a tuple in r; foreign key constraints requires that there should be a tuple t2 in S such that t1[FK] = t2[K] when t1[FK] is not null.

Here is complete schema of XIT database:



An instance XIT schema

Student

StudentID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5

Program

PID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

Department

DID	DName
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

Summary:

1. We have seen how mathematical relation has been adapted to represent databases.
2. A relation in relational model is set of tuples. A tuple is ordered list of values for respective attributes. A tuple can also be represented as set of attribute value pairs.
3. Defined: Key (candidate key), Super key, and Foreign Key.
4. Learned various types of Constraints: Domain, Key, Not Null, Entity Integrity, and Referential Integrity.
5. Foreign keys are means of recording "associations". Value in a foreign key refers to another tuple, mostly in the table but occasionally be in the same table.
6. FK specification tells following: (1) attribute(s) that is FK, and (2) attribute (and in which relation) referred by foreign key.
7. Referential integrity constraint tells us that value in a foreign key must refer to some existing tuple in referred relation.

Relational Database manipulation

1. Relational Algebra
2. Relational Calculus
3. Standard Query Language (SQL)

Relational Algebra

Relational algebra allows expressing queries as algebraic expression, where operands are relations. In example below, sigma (σ) is an operation, employee is operand relation, and logical expression is parameter to the operation.

$$\sigma_{\text{DNO = 4 AND SALARY > 50000}}\ (\text{employee})$$

This expression outputs tuples that member of employee relation, and value of salary attribute is > 50000 and value of dno attribute is equals to 4.

Relational algebra expressions have closure property that is result of an expression is a valid relation.

Relational algebra was introduced with original proposal of relational model.

There are definite set of operations defined on relations. Some operations are unary while others are binary.

Complex queries are expressed by sequencing multiple operations.

Relational Calculus

A query in relational calculus is expressed First Order Logic.

Above relational algebra query is expressed in tuple calculus as following.

$$\{ t \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{salary} > 50000 \text{ AND } t.\text{dno}=4) \}$$

The expression here can be understood as following: Outputs tuple variables t that (1) it is member of relation EMPLOYEE, and (2) t.dno = 4 and t.salary > 50000

Below is another query that outputs specific attributes of tuple variables t, where given logic expression evaluates to true. Here expression can be understood as following: t is tuple variable ranging over relation EMPLOYEE, d is tuple variable ranging over relation DEPARTMENT and d is such t.dno=d.dno and d.name is 'Research'.

$$\{t.\text{Fname}, t.\text{Lname}, t.\text{Address} \mid \text{EMPLOYEE}(t) \text{ AND } (\exists d)(\text{DEPARTMENT}(d) \text{ AND } d.\text{Dname}=\text{'Research'} \text{ AND } d.\text{Dnumber}=t.\text{Dno})\}$$

It may be noted that in calculus expressions are more declarative (rather than procedural) in nature; no concept of operation and their order.

Structured Query Language (SQL)

Structured Query Language (SQL) was not part of original proposal of Relational Model. Some people pronounce SQL as "SEQUEL".

Originally developed at IBM with System R. System R was first implementation of Relational Model in Early 70s.

SQL has evolved since then, and now a standard language for relational databases.

ANSI has published SQL-86, SQL-89, SQL-92, SQL-99, SQL-2003, SQL-2006, and SQL-2008, 2013 or so. SQL that discussed in most texts is SQL-99.

Despite ANSI SQL standard, RDBMS often do some kind of deviation to it. Postgres's SQL is probably most ANSI compliant.

SQL provides following categories of commands:

- Data Definition Language (DDL)
  - Used for defining database schema.
- Data Manipulation Language (DML)
  - Update commands
  - Querying commands
- Physical level commands
  - Used for improving performance of database.
- Transaction Control
  - Begin transaction, commit transaction, rollback etc
- Authorization
  - Grant permission to other users for access (read/update /delete, etc.)

Note: relational algebra and calculus do not allow expressing data definitions, and data update operations. SQL being the language that is used in practice, it provides full stack of operations.

SQL was initially much influenced by tuple relation calculus, however later updates incorporated most algebraic concepts as well. We will learn most of SQL in algebraic perspective.