Relational Database

Contents

1. *Introduction*
2. *Structure of Relational Databases*
3. *Database Schema*
4. *Keys*
5. *Schema Diagrams*
6. *Relational Query Languages*
7. *Relational Operations*
8. *Relational Algebra*
9. *Properties of Relations*
10. *Integrity Constraints*
11. *View*

1. Introduction

- The relational model was first proposed by E. F. Codd in his seminal paper 'A relational model of data for large shared data banks' (Codd, 1970).
- The relational model's objectives were specified as follows:
  - To allow a high degree of data independence. Application programs must not be affected by modifications to the internal data representation, particularly by changes to file organizations, record orderings, or access paths.
  - To provide substantial grounds for dealing with data semantics, consistency, and redundancy problems. In particular, Codd's paper introduced the concept of **normalized** relations, that is, relations that have no repeating groups.
  - To enable the expansion of set-oriented data manipulation languages.

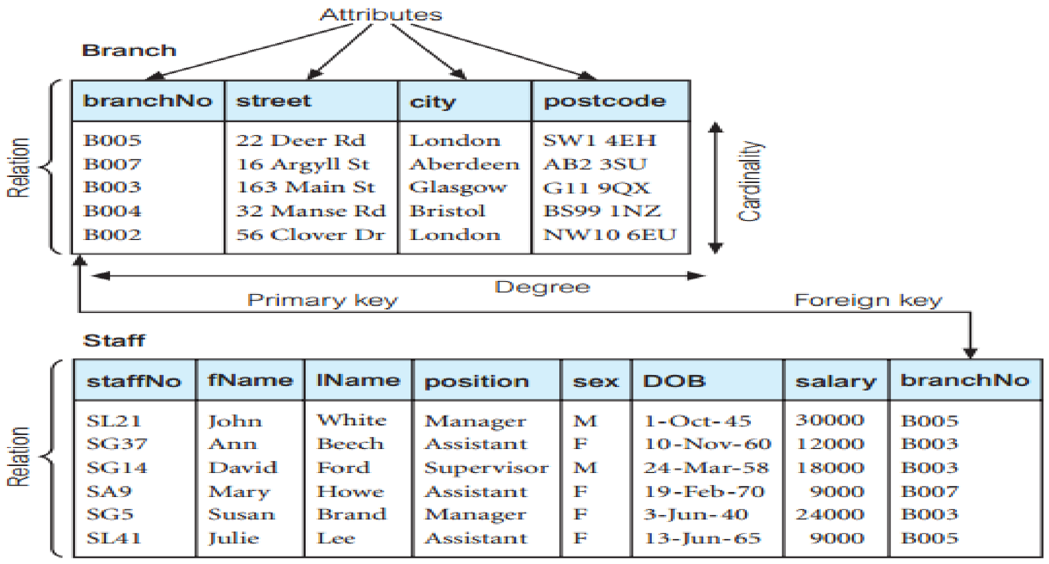
2. Structure of Relational Database

- A relational database consists of a collection of tables, each of which is assigned a unique name.
- The relational model is based on the mathematical concept of a **relation**, which is physically represented as a **table**.
- In general, a **row** in a table represents a relationship among a set of values.
- Since a table is a collection of such relationships, there is a close correspondence between the concept of table and the mathematical concept of relation, from which the relational data model takes its name.

- In mathematical terminology, a **tuple** is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an n-tuple of values, i.e., a tuple with n values, which corresponds to a row in a table.
- Thus, in the relational model the term relation is used to refer to a **table**, while the term **tuple** is used to refer to a row. Similarly, the term attribute refers to a column of a table.
- For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute. Thus, the domain of the salary attribute of the instructor relation is the set of all possible salary values.
- We require that, for all relations r, the domains of all attributes of r be **atomic**.
- A domain is **atomic** if elements of the domain are considered to be indivisible units.
- For example, suppose the table instructor had an attribute phone number, which can store a set of phone numbers corresponding to the instructor.
- The important issue is not what the domain itself is, but rather how we use domain elements in our database. Suppose now that the phone number attribute stores a single phone number. Even then, if we split the value from the phone number attribute into a country code, an area code and a local number, we would be treating it as a non-atomic value. If we treat each phone number as a single indivisible unit, then the attribute phone number would have an atomic domain.
- The **degree** of a relation is the number of attributes it contains while the **cardinality** of a relation is the number of tuples it contains.
- The **null** value is a special value that signifies that the value is unknown or does not exist. For example, suppose as before that we include the attribute phone number in the student relation. It may be that a student does not have a phone

number at all, or that the telephone number is unlisted. We would then have to use the null value to signify that the value is unknown or does not exist.

- We shall see later that null values cause a number of **difficulties** when we access or update the database, and thus should be eliminated if at all possible.



branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

3. Database Schema

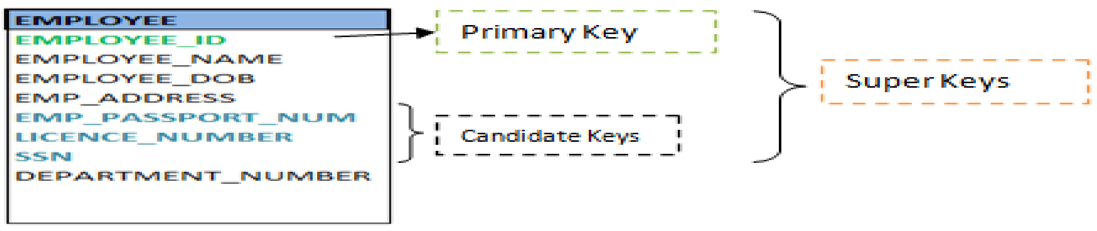
- When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time.
- The concept of a relation corresponds to the programming-language notion of a **variable**, while the concept of a relation schema corresponds to the programming-language notion of **type definition**.
- In general, a relation schema consists of a list of **attributes** and their corresponding **domains** while the concept of a relation instance corresponds to the programming-language notion of a **value of a variable**.

- The value of a given variable may **change with time**; similarly the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does **not generally change**.

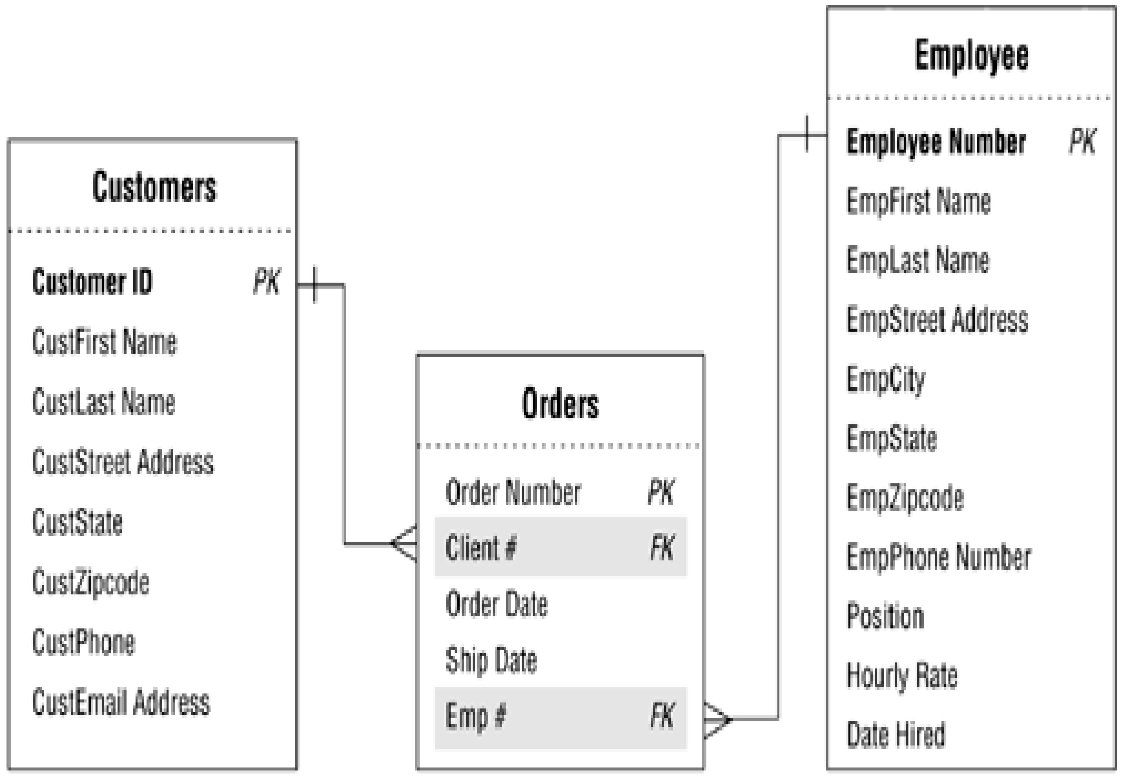
4. Keys

- We must have a way to specify how tuples within a given relation are distinguished. This is expressed in terms of their attributes. That is, the values of the attribute values of a tuple must be such that they can *uniquely identify* the tuple.
- In other words, no two tuples in a relation are allowed to have exactly the same value for all attributes. A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- For example, the *ID* attribute of the relation *Studennt* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey.
- The *name* attribute of *student*, on the other hand, is not a superkey, because several instructors might have the same name.
- Formally, let R denote the set of attributes in the schema of relation r. If we say that a subset K of R is a superkey for r, we are restricting consideration to instances of relations r in which no two distinct tuples have the same values on all attributes in K. That is, if t1 and t2 are in r and t1≠ t2, then t1. K≠ t2. K.
- A superkey may contain extraneous attributes. For example, the combination of ID and name is a superkey for the relation instructor. If K is a superkey, then so is any superset of K. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.

- It is possible that several distinct sets of attributes could serve as a candidate key.



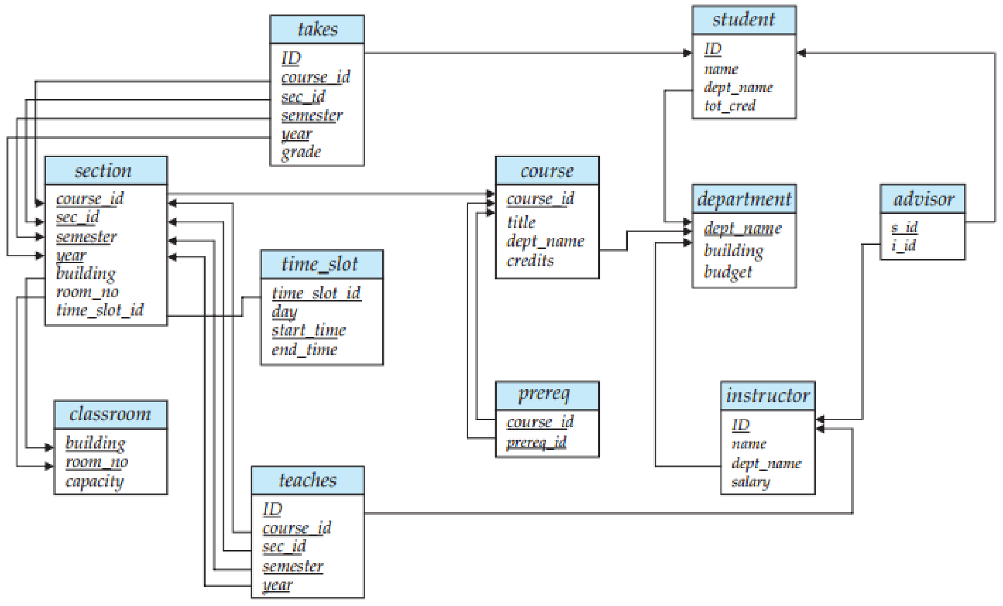
- We shall use the term primary key to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation. A key (whether primary, candidate, or super) is a property of the entire relation, rather than of the individual tuples.
- Any two individual tuples in the relation are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled.
- Primary keys must be chosen with care. As we noted, the name of a person is obviously not sufficient, because there may be many people with the same name.
- The primary key should be chosen such that its attribute values are never, or very rarely, changed.
- Example of primary to foreign key.



- **A referential integrity constraint** requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

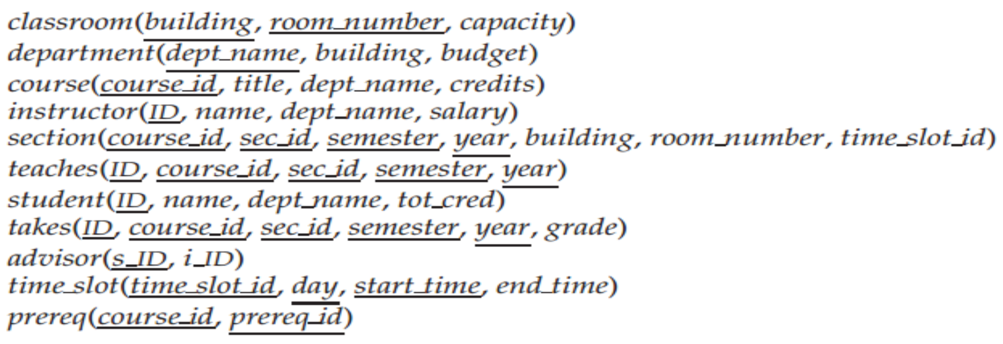
5. Schema Diagram

- A database schema, along with primary key and foreign key dependencies, can be depicted by schema diagrams. The schema diagram for a university organization shown in the figure below.
- Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.



6. Relational Query Languages

- A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either **procedural or nonprocedural**. In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result (C++, JAVA, Pascal). In a nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information (SQL, PROLOG, Visual Basic).



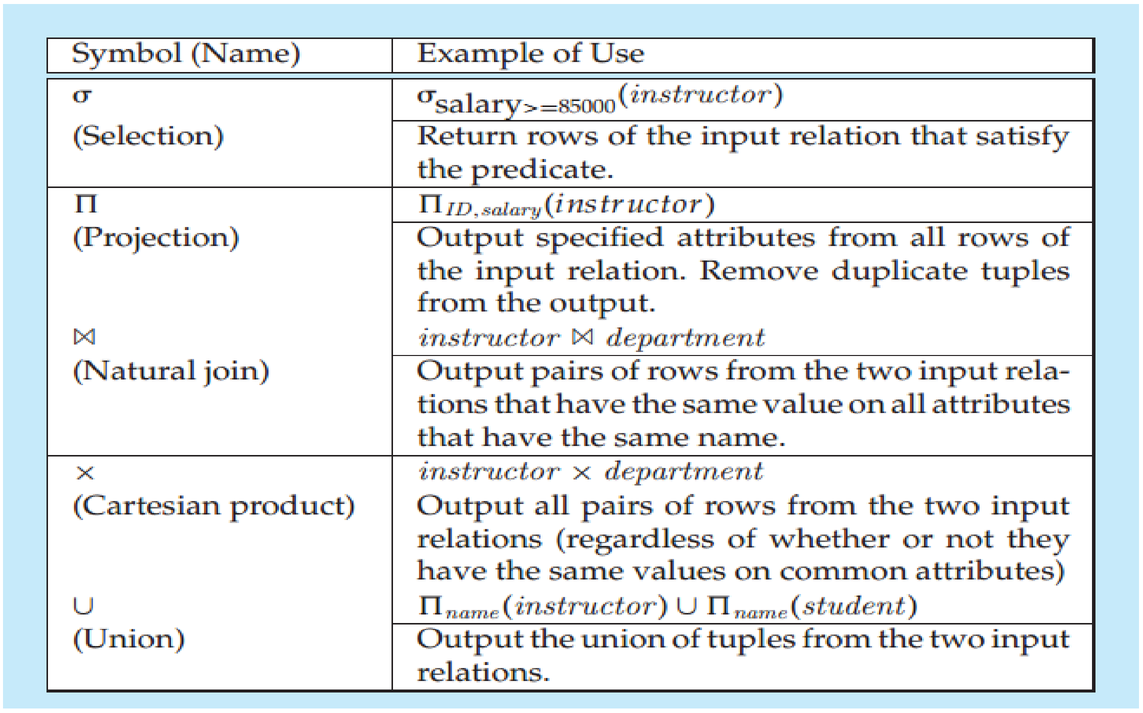
classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

7. Relational Operations

- All relational query languages provide a set of operations that can be applied to either a single relation or a pair of relations. These operations have the nice and desired property that their result is always a single relation.
- This property allows one to combine several of these operations in a modular way. Specifically, since the result of a relational query is itself a relation, relational operations can be applied to the results of queries as well as to the given set of relations.
- The frequent operation is to **select** certain attributes (columns) from a relation. The result is a new relation having only those selected attributes.
- The **join** operation allows the combining of two relations by merging pairs of tuples, one from each relation, into a single tuple.
- The *Cartesian product* operation combines tuples from two relations, but unlike the join operation, its result contains *all* pairs of tuples from the two relations, regardless of whether their attribute values match.
- The *union* operation performs a set union of two "similarly structured" tables (say a table of all graduate students and a table of all undergraduate students). For example, one can obtain the set of all students in a department. Other set operations, such as *intersection* and *set difference* can be performed as well.

8. Relational Algebra

- The relational algebra defines a set of operations on relations, paralleling the usual algebraic operations such as addition, subtraction or multiplication, which operate on numbers.
- Just as algebraic operations on numbers take one or more numbers as input and return a number as output, the relational algebra operations typically take one or two relations as input and return a relation as output.



Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$ Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID, salary}(instructor)$ Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$ (Natural join)	$instructor \bowtie department$ Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$ (Cartesian product)	$instructor \times department$ Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$ Output the union of tuples from the two input relations.

9. Properties of Relations

1) The relation has a **name** that is distinct from all other relation names in the relational schema;
2) Each cell of the relation contains exactly **one atomic** (single) value;

3) Each **attribute** has a distinct name;
4) The values of an attribute are all from the same **domain**;
5) Each tuple is **distinct**; there are no duplicate tuples;
6) The **order** of attributes has no significance;
7) The order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples.)

10. Integrity Constraints

- Since every attribute has an **associated domain**, there are constraints (called **domain constraints**) that form restrictions on the set of values allowed for the attributes of relations.
- In addition, there are two important integrity rules, which are constraints or restrictions that apply to all instances of the database. The two principal rules for the relational model are known as **entity integrity** and **referential integrity**.

10.1Null

- A null can be taken to mean the logical value '**unknown'**. It can mean that a value is **not applicable** to a particular tuple, or it could merely mean that no value has yet been supplied. Nulls are a way to deal with **incomplete or exceptional data**.
- However, a null is not the same as a zero numeric value or a text string filled with spaces; zeros and spaces are values, but a null represents the absence of a value. Therefore, nulls should be treated differently from other values.

- Some authors use the term ‘null value’, however as a null is not a value but represents the **absence of a value**, the term ‘null value’ is **deprecated**.

10.2 Entity integrity

- In a base relation, no attribute of a primary key can be null. By definition, a primary key is a minimal identifier that is used to identify tuples uniquely.
- This means that no subset of the primary key is sufficient to provide unique identification of tuples. If we allow a null for any part of a primary key, we are implying that not all the attributes are needed to distinguish between tuples, which contradicts the definition of the primary key.

10.3 General Constraints

- Additional rules specified by the users or database administrators of a database that define or constrain some aspect of the enterprise.
- It is also possible for users to specify additional constraints that the data must satisfy. For example, if an upper limit of 20 has been placed upon the number of staff that may work at a branch office, then the user must be able to specify this general constraint and expect the DBMS to enforce it. In this case, it should not be possible to add a new member of staff at a given branch to the Staff relation if the number of staff currently assigned to that branch is 20.

11. View

- In the relational model, a view is a **virtual** or **derived relation**: a relation that does not necessarily exist in its own right, but may be dynamically derived from one or more **base relations**.

- **A View** The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a *virtual relation* that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.
- A view is a relation that **appears to the user to exist**, can be manipulated as if it were a base relation, but **does not necessarily exist in storage** in the sense that the base relations do (although its definition is stored in the system catalog).
- The contents of a view are defined as a **query** on one or more base relations. Any **operations** on the view are automatically translated into operations on the relations from which it is derived.
- Views are **dynamic**, meaning that changes made to the base relations that affect the view are immediately reflected in the view.
- When users make permitted changes to the view, these changes are made to the underlying relations. In this section, we describe the purpose of views and briefly examine restrictions that apply to updates made through views.

11.1 Purpose of Views

- The view mechanism is desirable for several reasons:
    - It provides a **powerful and flexible security mechanism** by hiding parts of the database from certain users. Users are not aware of the existence of any attributes or tuples that are missing from the view.
    - It permits users to access data in a way that is **customized to their needs**, so that the same data can be seen by different users in different ways, at the same time.
    - It can **simplify complex operations** on the base relations. For example, if a view is defined as a combination (join) of two relations,

users may now perform more simple operations on the view, which will be translated by the DBMS into equivalent operations on the join.

11.2 Updating Views

- All updates to a **base relation** should be immediately reflected in all views that reference that base relation.
- Similarly, if a view is updated, then the underlying base relation should reflect the change. However, there are restrictions on the types of modification that can be made through views.
- We summarize below the conditions under which most systems determine whether an update is allowed through a view:
    - Updates are allowed through a view defined using a simple query involving a single base relation and containing either the primary key or a candidate key of the base relation.
    - Updates are not allowed through views involving multiple base relations.
    - Updates are not allowed through views involving aggregation or grouping operations. Classes of views have been defined that are **theoretically not updatable**, **theoretically updatable**, and **partially updatable**. A survey on updating relational views can be found in Furtado and Casanova (1985)