Database Driven: *Relational Database Theory*

Greg Ferguson
June 3, 2005

ICS 139W

Abstract

Relational database theory is a fairly young branch of computer science but a very important one. Database Management Systems (DBMS) allow for large volumes of data to be stored and accessed without a concern for how the data is stored. A database that is designed with the principles of relational database theory will have no data redundancy and will ensure that correct relationships are maintained. Database decomposition and Boyce-Codd Normal Form (BCNF) are the proper methods used in the design of a relational database.

1 Introduction

Relational database theory is the driving force behind Database Management Systems (DBMS). Database Management Systems allow the user to store and maintain information. The information that is stored can be related to other data. Relational database theory governs how the data is related. The user of a DBMS doesn't need to concern themselves with implementation details of how the data is physically stored. The user does need to know the data representation of a relational database system.

1.1 Data Representation

A *relation*, table, is the storage container of the relational database. A relation contains a set of *attributes*, columns, which describe which type of data is stored in the relation. Each attribute is associated with a *domain* which is the physical type of data that is to be stored. Typical domains include integer, string, and double. All data values that are associated with a given attribute share the same domain.

Relation				
A_1	A_2	…	A_n	Attributes
$t_1[A_1]$	$t_1[A_2]$		$t_1[A_n]$	
$t_2[A_1]$	$t_2[A_2]$		$t_2[A_n]$	Tuples
…				
$t_3[A_1]$	$t_3[A_2]$		$t_3[A_n]$	

Figure 1. This diagram shows the composition of a relation and how data cells can be accessed by attribute and tuple. [3]

A *relation schema* is what describes a given relation and contains information such as the relation name and set of attributes.

```
Person (SSN,FirstName,LastName)        (1)
```

The above relation scheme has a relation name of Person and a set of attributes which include SSN, FirstName, and LastName. A *database schema* contains a database name and a set of relation schemas.

Each relation contains a set of *tuples*, rows. Each tuple contains a data values for each attribute that is in the relation. As shown in figure 1, a data cell can be accessed by knowing the attribute and tuple. A

relation instance is a relation at a given moment in time; the set of tuples at this moment in time is known and remains unchanged.

1.2 Relational Concepts

Relational theory is closely related to set theory and the same rules that apply to sets also apply to relations. There are three important rules which govern relations.

- The ordering of tuples within a relation is irrelevant.
- Each tuple in a relation is distinct.
- The ordering of attributes within a relation is irrelevant. Therefore the order of data values within a tuple is irrelevant. Order can be ignored because data values are accessed by naming the attribute of the data value that is being accessed.

2 Integrity Constraints

Integrity constraints are restrictions placed on the tuples which can appear in a given relation. Integrity constraints constrain the data values which can appear in tuples.

2.1 Keys

Keys are used within relations to maintain integrity constraints. Every relation contains a set of attributes. A subset of the attributes within the relation can have the *unique identification property* if for every tuple in the relation no two tuples have the same data values for the given subset of attributes. Any subset of attributes which has the unique identification property is called a *superkey*. If the subset of attributes is the minimal set of attributes that guarantees the unique identification property than the subset is called a *key*.

Relation Instance 1		
SSN	FirstName	LastName
555-55-5555	John	Smith
444-44-4444	Joe	Thompson
Relation Instance 2		
SSN	FirstName	LastName
555-55-5555	John	Smith
444-44-4444	Joe	Smith

Figure 2. Two relation instances are shown. In relation instance 1 the set of attributes {SSN, FirstName, LastName} is a superkey because no two tuples have the same data value for a given attribute. In relation instance 2 the set of attributes {SSN, FirstName} is a superkey. For both relation instances the set of attributes {SSN} is a key because it is a minimal set. The set of attributes {SSN} is also a superkey because it guarantees the unique identification property for all possible relation instances.

As shown in figure 2, a subset of attributes can be a key for one relation instance but the same key may not be true for every relation instance. If a subset of attributes is a key for every possible relation instance than it is a *superkey*, primary key.

2.2 Functional Dependencies

Functional Dependencies (FDs) are an integrity constraint that is placed on a relation schema. Functional dependencies can be found by analyzing the requirements of the data within the database. Functional dependencies are always obeyed within the relation schema. Code segment 2 shows a function dependency for the Person relation.

```
SSN → FirstName, LastName                (2)
```

This FD tells us that the set of attributes {SSN} functionally determines the set of attributes

{FirstName, LastName}. A set of functional dependencies can be contained within a database schema. Code segment 3 shows a set of functional dependencies which describe the student database schema.

```
SSN → FirstName, LastName                (3)
courseID → courseTitle
SSN, courseID → grade
```

The database schema has a set of three functional dependencies. Each functional dependency describes a relation schema within the database schema. The database schema contains three relations. Code segment 4 shows the student database schema.

```
Person (SSN, FirstName, LastName)        (4)
Course (courseID, courseTitle)
Grades (SSN, courseID, grade)
```

In the grades relation it is said that the set of functional dependencies {SSN, courseID} functionally determines the set of functional dependencies {grade}.

A functional dependency can be derived from a set of functional dependencies in order to create a more succinct set of functional dependencies. Three rules govern the manipulation and derivation of functional dependencies.

- *Reflexivity* (Rule 1). Functional dependencies are reflexive; any attribute always can determine itself.
  X → X
- *Decomposition* (Rule 2). Functional dependencies can be decomposed. If a set of attributes determines a set of attributes than it also determines a subset of that set.
  If X→YZ then X→Y.

F is a set of functional dependencies.
F {E→G,BGE→DH,AB→B,G→DE}
Does the following FD hold? ABG→E

1) G→G	from rule 1
2) G→DE	in F
3) G→GDE	from rule 3 by 1, 2
4) ABG→ABG	from rule 1
5) ABG→GDEAB	from rule 3 by 3,4
6) ABG→E	from rule 2 by 5

Figure 3. A derivation of the functional dependency ABG→E from the given set of functional dependencies, F, is shown.

- *Transitivity* (Rule 3). Multiple functional dependencies can be decomposed. If a set of attributes determines a second set of attributes and a subset of the second set of attributes determines a third set of attributes than the first set of attributes functionally determines the second and third set of attributes.
  If X→YW, W→Z then X→YWZ.

Figure 3 shows a functional dependency being derived from a set of functional dependencies. It is derived using the three rules governing functional dependency manipulation.

3 Operations on Relations

Operations can be preformed on relations in order to obtain desired data results. All operations that are preformed on relations take one or more relations as arguments and produce relations as output. Because of this property multiple operations can be combined to produce one output. Several operations exist that can be preformed on relations.

- Selection
- Projection
- Join
- Rename

Because relations contain sets of tuples, operations that can be preformed on sets can also be preformed on relations.

- Union
- Intersection
- Difference

A restriction is placed on the operations that are derived from set theory. When performing a union, intersection, or difference the relations that are being used as the operands must contain the same set of attributes.

3.1 Selection

A *selection*, σ, is used to select a set of tuples from a relation based on a selection criterion.

$\sigma_{\text{FirstName = "John"}}$ (Person) (5)

Results:

SSN	FirstName	LastName
555-55-5555	John	Doe
444-44-4444	John	Thompson

The statement in code segment 5 selects every tuple where FistName is equal to the string "John" from the Person relation instance in Figure 4. A possible result to the code segment is also shown.

3.2 Projection

A *projection*, π, is used to select the attributes that will appear in the result relation.

$\pi_{\text{FirstName}}$ (Person) (6)

Results:

FirstName
John
John

Person		
SSN	FirstName	LastName
555-55-5555	John	Smith
444-44-4444	John	Thompson

Enrollment	
SSN	Course
555-55-5555	ICS 131
555-55-5555	ICS 141

Figure 4. Two relation instances are shown.

The statement in code segment 6 selects the FirstName attribute from the Person relation instance, in Figure 4, to be shown in the set of result attributes. A possible result to the code segment is also shown.

3.3 Natural Join

A *natural join* can be used to compare the values associated with an attribute in one relation to the values associated with an attribute in another relation.

Person JOIN Enrollment on SSN (7)

Results:

SSN	FirstName	LastName	Course
555-55-5555	John	Doe	ICS 131
555-55-5555	John	Doe	ICS 141

A natural join of the two relation instances shown in figure 4 is shown in code segment 7. The SSN attribute only occurs once in the resulting set of attributes because no two duplicate attributes can appear in the same set.

3.4 Rename

A *rename*, ρ, is used to rename one of the attributes that will appear in the result relation.

$\rho_{FirstName \rightarrow fName}$ (Person) (8)

Results:

SSN	fName	LastName
555-55-5555	John	Doe
444-44-4444	John	Thompson

The statement in code segment 8 renames the FirstName attribute in the Person relation, which is found in Figure 4, to fName. A possible result set to the code segment is also shown.

3.5 Union, Intersection and Difference

Union, intersection and difference are used as they are in set theory. Each of these operations can be used on relations, sets of tuples, with like attributes.

4 Boyce-Codd Normal Form

A relation scheme which is in Boyce-Codd Normal Form[1] (BCNF) has a minimal set of functional dependencies and a set of primary keys. If all relations within a database scheme are in BCNF than the database scheme is said to be in BCNF. In order to obtain a minimal set of functional dependencies it may be necessary to manipulate the set of functional dependencies using the three manipulation rules or using known algorithms.

Typical DBMS systems do not enforce functional dependencies, they only enforce keys. However, the enforcement of keys within a BCNF database schema is enough to ensure that functional dependencies are also being enforced.

[1] BCNF covers one to one relationship and one to many relationships within the data set. BCNF is most like the third normal form.[2]

```
Relational Operations
      πSSN,LastName,Courses (
      ρLastName → lName (
      (σFirstName = "John" (Person))
      JOIN Enrollment on SSN))

SQL Representation
      SELECT SSN,LastName as lName,Course
      FROM Person
      INNER JOIN Enrollment on
      Enrollment.SSN = Person.SSN
      WHERE FirstName = "John";
```

Figure 5. A complex relational operation is shown with its SQL equivalent.

5 Decomposing into BCNF

It is useful to create a non-redundant set of functional dependencies so that data is stored in the minimum number of places and isn't subject to update/delete anomalies. Decomposition is the accepted method of finding a non-redundant set of functional dependencies is called *decomposition.* The resulting non-redundant set of functional dependencies is:

- *L-minimum.* A functional dependency that is L-minimum contains the least number of attributes on the left side of the function dependency.
- *R-minimum.* A functional dependency that is r-minimum contains the least number of attributes on the right side of the functional dependency.
- *Non-circular.* A set of functional dependencies that is non-circular contains no circularity within its functional dependencies.

```
Circular Set of FDs                    (9)
F {A→BC, B→A}

Non-circular Set of FDs
F {A→BCD, B→A}
```

Code segment 9 shows a circular set of functional dependencies and a non-circular set of functional dependencies. An algorithm exists to create the LR-minimum non-circular set of functional dependencies. A set of functional dependencies that is LR-minimum non-circular will have no data redundancy.

OPERATION	*SQL*
$\sigma_{\texttt{FirstName = "John"}}$ (Person)	SELECT * FROM Person WHERE FirstName = "John"
$\pi_{\texttt{FirstName}}$ (Person)	SELECT FirstName FROM Person
Person JOIN Enrollment on SSN	SELECT * FROM Person INNER JOIN Enrollment on Enrollment.SSN = Person.SSN
$\rho_{\texttt{FirstName} \rightarrow \texttt{fName}}$ (Person)	SELECT FirstName as fName FROM Person

Table 1. Operations on relations and their SQL equivalent are presented.

6 Structured Query Language

Structured query language, SQL[5], is used by users of a DBMS in order to extract data from the database. Using SQL a user can express any of the five operations on relations that were previously mentioned in this paper. Table 1 shows operations that can be preformed on relations and their SQL equivalent. Figure 5 shows a complex relational operation and its SQL representation.

7 Conclusion

An efficient database schema can be achieved by using the principles of relational database theory. A LR minimal non-redundant set of functional dependencies must be found. BCNF relation schemas can be created from the resulting set of functional dependencies. The BCNF database schema will contain each of the resulting BCNF relation schemas.

The resulting BCNF database schema will have no data anomalies and will contain no false relationships [4]. The user of the database can use SQL to perform operations on the relations within the database and the resulting relations will contain only correct data relationships.

Database systems that do not use relational database theory do exist as an open problem in computer science today. Object oriented database are currently being studied [1] but a good solution to the problem has yet to be discovered. The relational model is the best solution available today.

Acknowledgement

Many of the ideas included in this paper were covered in the lectures of Craig A. Rich of Cal Poly Pomona, spring 2004. Through his lectures I was able to increase my understanding of database systems and relational database theory.

References

[1] Atkinson, Malcolm et. al. The Object Oriented Database System Manifesto.

[2] Janert, Phillipp K. Software Project Consultant. Practical Database Design, Part 2. 27 April, 2004. DevX. <http://www.devx.com/ibm/Article/20859>

[3] Rich, Craig A. Lecture Material. CS435. Cal Poly Pomona. Spring 2004. <http://www.csupomona.edu/~carich/classes/cs435>

[4] Ullman, Jeffrey D. et. al. A First Course in Database Systems. Second Edition. Prentice Hall. New Jersey. 2002. ISBN 0-13-035300-0.

[5] Widom, Jennifer et al. Set-Oriented Production Rules in Relational Database Systems. IBM Almaden Research Center. Proc. of 1990 ACM-SIGMOD Conference, p. 259-270. <ftp://db.stanford.edu/pub/papers/rule-language.ps>