The Relational Model and Relational Algebra

Objectives

- Understand why the relational model is so important
- Be able to describe the essential features of the relational model
- Understand and use the various concepts of integrity that apply to the relational model
- Map an EER model into a relational model
- Read and understand the meaning of a query expressed in relational algebra
- Express a relational database query in relational algebra
- Understand how a relational algebra expression interrogates a relational database

Introduction

- The relational model is the structure of the actual software in which we build our database having done the designing using the EER model.
- The relational database is the most popular type in existence.
- It is built on a mathematical foundation that makes everything to do with it susceptible to mathematical-style proofs of correctness.
- It was first introduced by Ted Codd in 1970 and use set theory and predicate logic to form its constructs.
- An SQL query says what you want but doesn't tell the database how to go about actually performing the query.

Structures of Relational Model

- In the relational model, all data is logically structured within **Relations** (which you can think of as tables).
- Each relation has a name and is made up of named **Attributes** (these are the table columns) of data.
- Each **Tuple** (table row) contains one value per attribute.
- A great strength of the relational model is this simple logical structure.

Relation

- A relation is a table with rows and columns
- Relations are used to hold information about the objects or entities to be represented in the database.
- Relations are only the logical structure of the database and not the physical structure of the database.

Attributes

- Properties characterizing an object or entity.
- Informal equivalent of attribute is column or field in a conventional file.
- For example, Student No., Student Name, City and Postal Code.
- Attributes can appear in any order and the relation will still be the same.

Domain

- Set of allowable values for one or more attributes.
- The set of scalar values which are all of the same type is called a domain.
- Domains are pools of values from which actual values appearing in attributes are drawn.

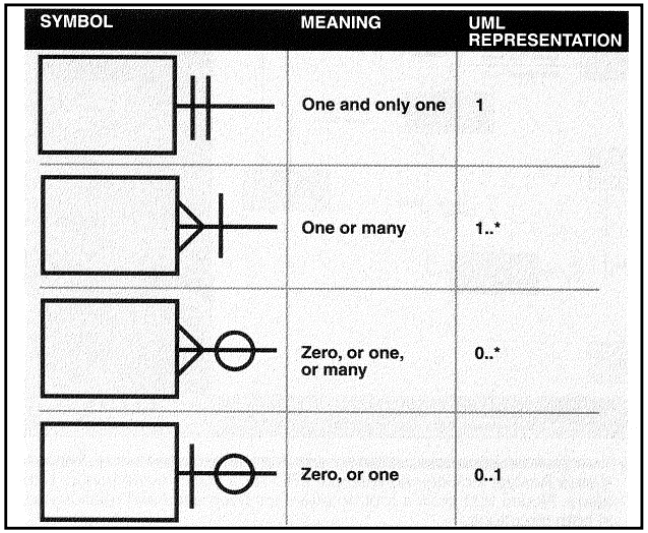
Tuple

- A tuple is a row of a relation
- It consists of a full set of attributes of a relation that characterizes an entity instance.
- Tuples can appear in any order and the relation will still remain the same.
- The tuples are called the extension (or state) of a relation, which changes over time.

Cardinality

- Number of tuples in a relation.
- For a relation, cardinality changes with time as tuples are added or deleted.

Cardinality



Degree

- Number of attributes of a relation schema.
- Degree does not change with time; changes in relation schema may change the degree.

Characteristics of a Relational Table

- A table is perceived as a two-dimensional structure composed of rows and columns.
- Each row (tuple) represents a single entity within the entity set.
- Each column represents an attribute, and each column has a distinct name.
- Each row/column intersection (cell) represents a single (atomic) data value.
- Each table must have an attribute or combination of attributes (called a primary key) the value(s) of which uniquely identifies each row.

Characteristics of a Relational Table

- All values in a column must conform to the same data format.
- Each column has a specific range of values known as the attribute domain.
- Each row carries information describing one entity occurrence.
- The order of the rows and columns is immaterial to the database

Example

- Since each cell should contain only one value, it would be illegal to store two IdNo for a single student in a single cell.
- In other words, relations do not contain repeating groups.
- A relation that satisfies this property is said to be normalized or in the first normal form.

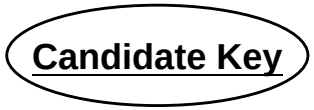
STUDENT	Name	IdNo	DateOfBirth	Address	Telephone
	Dave Garlick	98-45-1234	21/12/77	4711 Fuschia Lane	022 6754 2387
	Kwan Kai Tai	99-53-3476	21/12/77	321 Wellborn Drive	null
	Mary Santiago	98-11-6643	12/7/65	4711 Fuschia Lane	024 9823 0853
	Jerry Cooper	99-24-7834	31/5/75	763 Carlton Avenue	0456 78239

Superkey

- If we can find a subset of the attributes of a relation, call the subset K, with the property that at any given time, no two tuples of relation R can have the same combination of values for these attributes then that set is called a **Superkey**.
- Every relation has at least one superkey, the set of all its attributes!
- If we have a superkey with the additional property that removing any attribute from the set K will leave a set that is no longer a superkey, then K is called a key.
- A key is a minimal superkey.

Candidate Key

- Candidate key is an attribute (or combination of attributes) that uniquely identifies each instance of an entity type



Primary Key

- Primary Key is a candidate key that has been selected as the identifier for an entity type.

Example

- We would choose IDNo as the key for relation STUDENT
  - Although the values of all of the attributes are actually distinct, it is possible for two different students to share a birthday, an address or even a name.
- A particular IDNo will be given to just one student.
- Relation will have more than one key. When this happens we call each of the keys a **Candidate Key**.

Name	*IDNo*	*DateOfBirth*	*Address*
Dave Garlick	98-45-1234	21/12/77	4711 Fuschia Lane
Kwan Tai Tai	99-53-3476	21/12/77	321 Wellborn Drive
Mary Santiago	98-11-6643	12/7/65	4711 Fuschia Lane
Jerry Cooper	99-24-7834	31/5/75	763 Carlton Avenue

Example

- Both DeptName and DeptNo are keys, so they are each candidate keys:
- We then choose one of the candidate keys whose values will be used to identify tuples in the relation. This is called the **Primary Key**. We normally denote the primary key by underlining.

DEPARTMENT

DeptName	*DeptNo*	*OfficePhoneExt*
Computer Science	111	4211
Accountancy	666	3125
Economics	999	3125

Concept of NULL values

- NULLS are intended to deal with missing information.
- A NULL does not mean a zero, a blank, or a space. Most of the time it means “not defined” or “not applicable”
- Integrity rules will define which attributes can take NULL and which cannot.

Relational Integrity Constraints

- **Constraints** are conditions that must hold on all valid relation instances.
- There are three main types of constraints:
  - Key Constraints
  - Entity Integrity Constraints
  - Referential Integrity Constraints

Entity Integrity Constraint

- The **Entity Integrity Constraint** says that no primary key value can be null.
- This follows from the fact that we use the primary key to identify individual tuples in a relation.
- If null values were allowed then we might not be able to do the identification.

Referential Integrity Constraint

- The **Referential Integrity Constraint** says that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Foreign Key

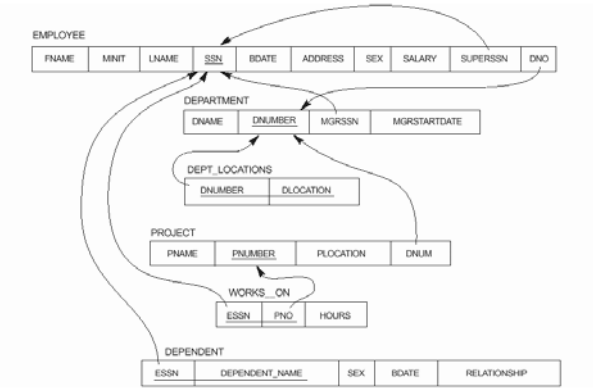
- A **Foreign Key** (FK) is an attribute or set of attributes of a relation R1 whose values are required to reference those of the primary key of some relation R2 by satisfying the following two rules:
  - The attributes in FK have the same domain(s) as the primary key attributes PK of R2
  - A value of FK in a tuple of R1 either occurs as a value of PK for some tuple in R2 or is null.

Why Constraints Important?

- One reason is that they define what we can and cannot do in manipulating the data in our database.
- If we concentrate on update operations (insert, delete and modify), then the constraints tell us what operations can be performed and which are illegal (in the sense that they violate the constraints).

Example

- Referential Integrity Constraints displayed on the COMPANY relational database schema diagram.



Constructing a Relational Database using an EER Model

- Once we have built an EER (or just an ER) model for our application we can map that design into a relational model by using a straightforward algorithm.

Rule 1

- For each regular entity type, E, create a relation including all simple attributes of E.
- Choose one of the key attributes of E as the primary key.

Rule 2

- For each weak entity type, W, with owner entity type E, create a relation including all simple attributes of W and the primary key of relation corresponding to E (called the owner relation).
- The primary key is the combination of the primary key of the owner relation and the partial key of W.

Rule 3

- For each binary 1:1 relationship R, take the relationships corresponding to the entity types participating in R, and post the key of one as a foreign key in the other.

Rule 4

- For each non-weak binary 1:N relationship R, post as a foreign key into the relation corresponding to the N-side, the primary key of the relation corresponding to the entity on the 1-side.

Rule 5

- For each binary M:N relationship R, create a new relation including any simple attributes of R and having as foreign keys, the keys of the relations corresponding to the entities of each side of R. These posted keys form the primary key.

Rule 6

- For each multivalued attribute A, create a relation that includes an attribute corresponding to A plus the primary key, K, of the relation that corresponds to the entity or relationship that has A as an attribute.
- The primary key is the combination of A and K.

Rule 7

- For each n-ary relationship R, n>2, create a relation including, as foreign keys, the primary keys of all the relations that represent the participating entity types and also any simple attributes of R itself.
- The primary key is usually the combination of the foreign keys. However, if the participation constraint of one of the entity types has max = 1, then the primary key can be the foreign key of the relationship corresponding to that entity type.

Rule 8

- For each superclass/subclass relationship: (We use Attrs(R) to denote the attributes of relation R and PK(R) to denote the primary key of R.)
- Convert each specialization with m subclasses {S1,S2,..... Sm} and (generalized) superclass C,where the attributes of C are {k, a1,. . ., an} and k is the (primary) key, into relation schemas using one of the four following options:

Rule 8 – Option A

- Create a relation L for C with attributes Attrs(L) = {k, a_1, …….,a_n} and PK(L) = k.
- Create a relation L_i for each subclass S_i, $1 \le i \le m$, with the attributes Attrs(L_i) = {k} È {attributes of S_i} and PK(L i) = k.

Rule 8 – Option B

- Create a relation L_i for each subclass S_j, $1 \le i \le m$, with the attributes $Attrs(L_i)$= {attributes of S_i} È {k, a_1,....,a_n} and $PK(L_i) = k$.

Rule 8 – Option C

- Create a single relation L with attributes Attrs(L) = {k, a_1,.....,a_n} È {attributes of S_1} È……..È {attributes of S_m} È {t} and PK(L) = k.
- This option is for a specialization whose subclasses are disjoint, and t is a **type** attribute that indicates the subclass to which each tuple belongs, if any.
- This option has the potential for generating a large number of null values.

Rule 8 – Option D

- Create a single relation schema L with attributes Attrs(L) = {k, a_1,.....a_n} È {attributes of S_1} È.....È {attributes of S_m} È {t_1,t_2,.....t_m} and PK(L) = k.
- This option is for a specialization whose subclasses are overlapping (not disjoint), and each t_i $1 \le i \le m$, is a Boolean attribute indicating whether a tuple belongs to subclass Si.

Remark

- For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a **Surrogate Key**, when creating a relation to correspond to that category.

Relational Algebra

- Any implementation data model must include a set of operations that we can use to manipulate the data.
- Relational Algebra is a basic set of such operations and the use of these operators will allow the user to specify any basic retrieval request.
- An important point to note is that the result of a retrieval will be another relation, so that further processing can be performed.
- This means that we can nest relational algebra operators to permit complex processing to take place.

Relational Algebra Operation

- Because the relational operators tell us what to do and how to do it, we say that this a procedural query language.
- The relational algebra operations are usually divided into two groups.
  - One groups includes set operations from mathematical set theory; Set operations include union, intersection, difference and cartesian product
  - The other group consists of operations developed specifically for relational databases; These include select, project and join.

Example – University Database

STUDENT

Name	IDNo.	DateOf Birth	Registered	Address
Dave Garlick	98-45-1234	21/12/77	1998	4711 Fuschia Lane
Kwan Kai Tai	99-53-3476	10/10/78	1999	321 Wellborn Drive
Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way
Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue

DEPARTMENT

DeptName	DeptNo	OfficePhoneExt
Computer Science	111	4211
Accountancy	666	2114
Economics	999	3125

GRADE

StuIDNo	UnitCode	Grade
98-45-1234	AB123	A
99-53-3476	RW345	C
98-11-6643	RW345	D
99-24-7834	AB123	F
98-45-1234	RW345	B
98-11-6643	CD678	E
99-24-7834	CD678	A
98-11-6643	AB123	D
99-24-7834	GB987	F
98-11-6643	GB987	A

UNIT

UnitCode	DeptCode
AB123	111
RW345	111
CD678	999
GB987	666

Relational Operations – Select

- Selects horizontal subset (tuples) of the relation that satisfies the condition.
- Notation:
  - $\sigma_{<condition>}$(relation-name)
- Syntax:
  - select < relation-name> where <condition>

Example

- $\sigma_{\text{Registered} > 1988}(\text{Student})$
- select Student where Registered >1998

STUDENT	Name	IDNo	DateOfBirth	Registered	Address
	Kwan Kai Tai	99-3-3476	10/10/78	1999	321 Wellborn Drive
	Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue

Example

- $\sigma_{(\text{Registered} > 1988)\ \text{AND}\ (\text{IdNo} \neq 99\text{-}53\text{-}3476)}(\text{Student})$
- select Student where (Registered > 1998) AND (IDNo <> 99-53-3476)

STUDENT	Name	IDNo	DateOfBirth	Registered	Address
	Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue

Relational Operations – Project

- It takes a vertical subset of a relation; that is, it takes certain columns from the table and discards the others.
- Notation:
  - $p_{\text{<attribute-list>}}(\text{relation-name})$
- Syntax:
  - **project** < relation-name> **over** <(attribute 1, attribute,……..)>

Example

- $\pi_{\text{Name}}(\text{Student})$
- project Student over Name

STUDENT	Name
	Dave Garlick
	Kwan Kai Tai
	Mary Santiago
	Jerry Cooper

Complex Example

- We will want to apply several relational algebra operations, one after another.
- We can do this either by writing the combined expression as one big complicated expression or by using intermediate steps in which these intermediate relations are given names.
- Suppose we wanted to find the ID numbers of those students who were born after 1st of January 1976:
  - $\pi_{IDNo}(\sigma_{DateOfBirth>1/1/76}(Student))$
  - $\sigma_{DateOfBirth>1/1/76}(Student) \rightarrow Temp$
    $\pi_{IDNo}(Temp) \rightarrow Result$

Set Theoretic Operations

- Binary operations from mathematical set theory:
  - **UNION** ($R_1 \cup R_2$)
  - **INTERSECTION** ($R_1 \cap R_2$)
  - **SET DIFFERENCE** ($R_1 - R_2$)
  - **CARTESIAN PRODUCT** ($R_1 \times R_2$)

Union

- Combines two relations of the same types.
- The result is a relation that includes all tuples that are in either of the two original relations.
- Notation:
  - $R \cup S$
- Syntax:
  - <relation 1> **union** < relation 2>

Intersection

- Finds the common tuples in two relations.
- Notation:
  - $R \cap S$
- Syntax:
  - < relation 1> **intersect** < relation 2>

Difference

- Finds all the tuples, which are in the first relation but not the second.
- Notation:
  - R – S
- Syntax:
  - < relation 1> **difference** <relation 2>

Example

- Suppose we want to find all students who have got a grade A in some unit or were born after 1st July 1977.
  - $\sigma_{DateOfBirth > 1/7/77}(Student) \rightarrow Temp1$
    $\pi_{IDNo}(Temp1) \rightarrow Temp2$
    $\sigma_{Grade=A}(Temp2) \rightarrow Temp3$
    $\pi_{StuIDNo}(Temp3) \rightarrow Temp4(IDNo)$
    $Temp2 \cap Temp4 \rightarrow Result$

RESULT	IDNo
	98-45-1234
	99-24-7834
	98-11-6643
	99-53-3476

- Notice we have renamed the attribute from StuIDNo to IDNo to make Temp 2 and Temp4 union compatible.

Example

- Now suppose we wanted to find all students who have both got a grade A in some unit and were born after 1st July 1977. We could proceed as before, except that the last line would become:
  - $Temp2 \cup Temp4 \rightarrow Result$

RESULT	IDNo
	98-45-1234

The JOIN operation

- This operation is used to combine related tuples from two relations into single tuples in the new relation.
- Notation:
  - $R \rhd\lhd_{R.a\theta S.b} S$
- Syntax:
  - join <R> and <S> where <R. Attribute a is compared to S. Attribute b>
- The Greek letter θ (theta) is used to stand for “is compared to”. That comparison could be less than, greater than, equals, etc.

Theta Join

- Similar to a CARTESIAN PRODUCT followed by a SELECT.
- The condition c is called a join condition.
  - $R_1(A_1, A_2, ..., A_m) \triangleright\triangleleft_c R_2(B_1, B_2, ..., B_n) \rightarrow R(A_1, A_2, ..., A_m, B_1, B_2, ..., B_n)$

Example – Equijoin

- Student $\triangleright\triangleleft_{\text{Student.IDNo = Grade.StuIDNo}}$ Grade
- **join** Student **and** Grade **where** Student.Idno = Grade.StuIDNo

Name	IDNo	DateOfBirth	Registered	Address	StuIDNo	UnitCode	Grade
Dave Garlick	98-45-1234	21/12/77	1998	4711 Fuschia Lane	98-45-1234	AB123	A
Dave Garlick	98-45-1234	21/12/77	1998	4711 Fuschia Lane	98-45-1234	RW345	B
Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	99-24-7834	AB123	F
Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	99-24-7834	CD678	A
Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	99-24-7834	GB987	F
Kwan Kai Tai	99-53-3476	10/10/78	1999	321 Wellborn Drive	99-53-3476	RW345	C
Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	98-11-6643	RW345	D
Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	98-11-6643	CD678	E
Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	98-11-6643	AB123	D
Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	98-11-6643	GB987	A

Example – Natural Join

- Grade → Temp (IDNo, UnitCode, Grade)
  Student * Temp → Result

RESULT	Name	IDNo	DateOfBirth	Registered	Address	UnitCode	Grade
	Dave Garlick	98-45-1234	21/12/77	1998	4711 Fuschia Lane	AB123	A
	Dave Garlick	98-45-1234	21/12/77	1998	4711 Fuschia Lane	RW345	B
	Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	AB123	F
	Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	CD678	A
	Jerry Cooper	99-24-7834	31/5/75	1999	763 Carlton Avenue	GB987	F
	Kwan Kai Tai	99-53-3476	10/10/78	1999	321 Wellborn Drive	RW345	C
	Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	RW345	D
	Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	CD678	E
	Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	AB123	D
	Mary Santiago	98-11-6643	12/7/65	1998	2376 Rodeo Way	GB987	A

Example – DIVISION

- If we wanted to find the student numbers of those students who have grades for all the units. To do this we could find a relation containing student ID numbers and unit codes.

R	StuIDNo	UnitCode
	98-45-1234	AB123
	99-53-3476	RW345
	98-11-6643	RW345
	99-24-7834	AB123
	98-45-1234	RW345
	98-11-6643	CD678
	99-24-7834	CD678
	98-11-6643	AB123
	99-24-7834	GB987
	98-11-6643	GB987

S	UnitCode
	AB123
	RW345
	CD678
	GB987

Now if we divide R by S, the effect will be to pick out only those values of StuIDNo in R which match all the values of UnitCode in S. In other words:

R divided by S	StuIDNo
	98-11-6643