Relational Model - Concepts

pm_jat @ daiict

Relational Model - Concepts

- What is a relation?
- Relation schema and relation
- Constraints on a relation, or some constraints may involve more than one relations

Relational Model Concepts

- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper:
  - "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970

- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

Relational Model Concepts

- In relational model, database is a set of *relations (or tables)*.
- Origin of relational model is mathematical *relations*.
- Therefore, let us attempt to relate a relation here with mathematical relation

What is a *relation* in Math

- Suppose you have two sets A, and B; in math a *relation* is defined as **`{(a,b) | a in A, b in B}`**

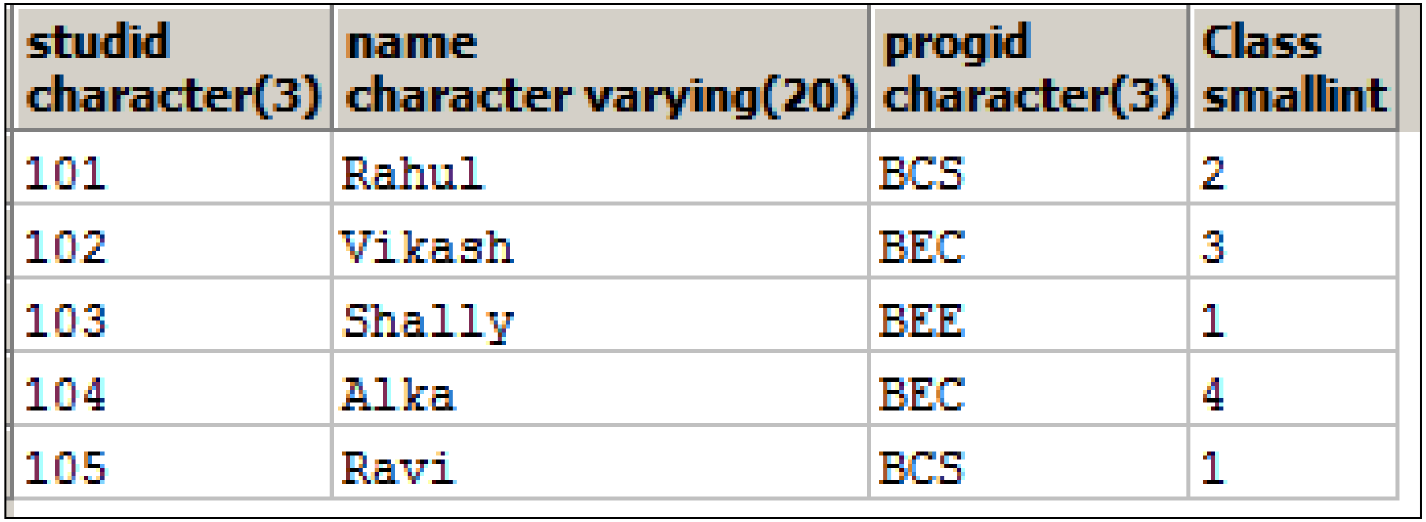
- Similarly from three sets A, B, and C, we have a relation as **`{(a,b,c) | a in A, b in B, c in C}`**

- For second case, we can also say that a relation is sub set of cross product of A, B, and C!

- A relation can also be defined as set of *n-tuples*, where values are drawn from respective sets. Recall that *n-tuple* is a ordered list.

Relation in Relational Model

- In relational model we introduce a notion of “attribute”, and each value in the tuple represents a “value for that attribute”.
- “A relation can also be defined as set of *n-tuples*, where values are drawn from ~~respective sets~~ respective attribute domain.



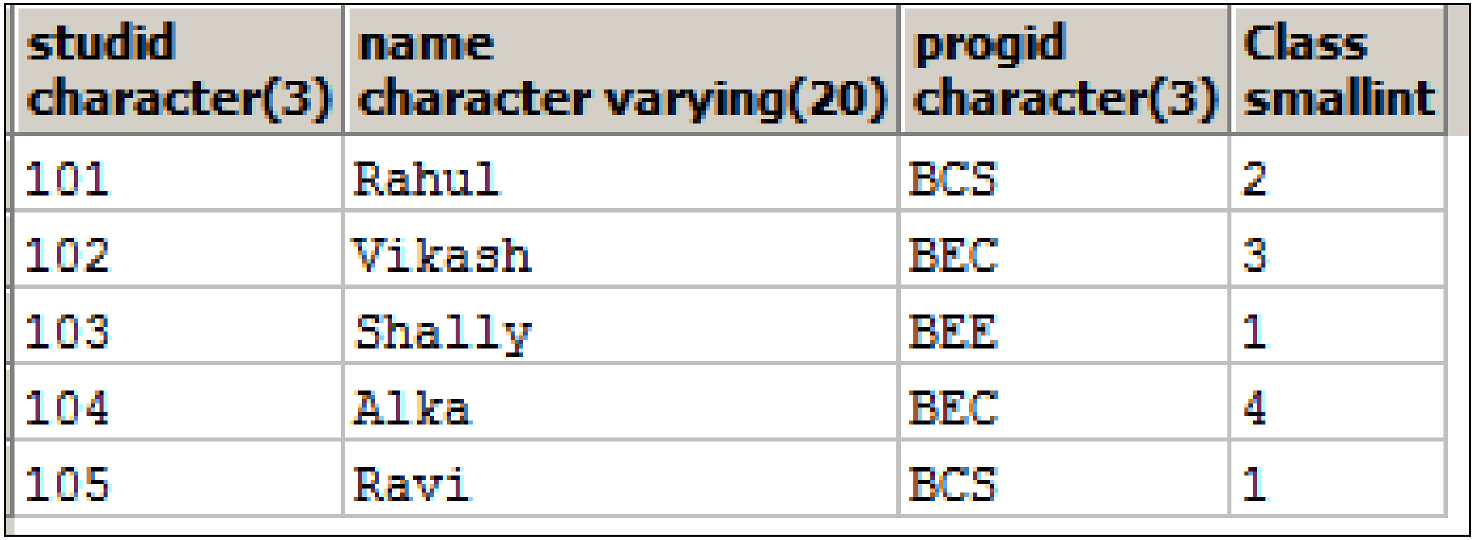
studid character(3)	name character varying(20)	progid character(3)	Class smallint
101	Rahul	BCS	2
102	Vikash	BEC	3
103	Shally	BEE	1
104	Alka	BEC	4
105	Ravi	BCS	1

Relation in Relational Model

- A relation R in the relational model is very similar to its counterpart in mathematics.

- A *relation* in relational model can be described as-
  - A relation (instance) r is set of tuples - ordering of tuple is immaterial
  - Each element in the tuple is ordered list of values (for respective attribute) drawn from respective attribute domain.
  - all tuples are distinct from one another "in content".

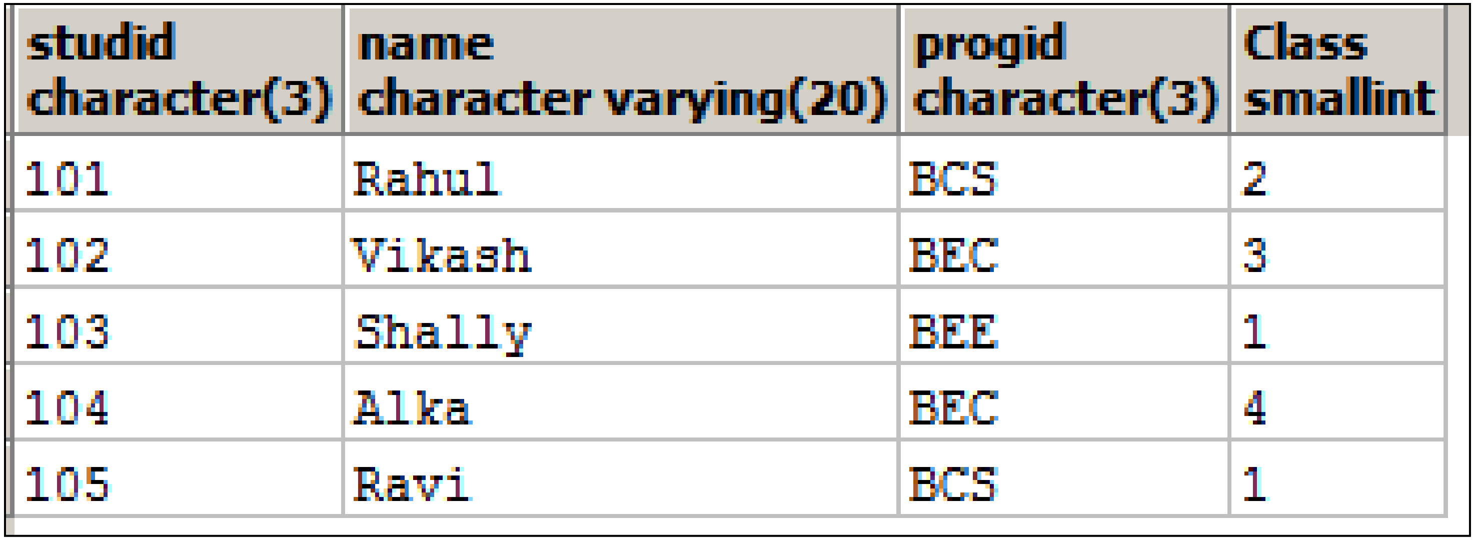
Relation in Relational Model



studid character(3)	name character varying(20)	progid character(3)	Class smallint
101	Rahul	BCS	2
102	Vikash	BEC	3
103	Shally	BEE	1
104	Alka	BEC	4
105	Ravi	BCS	1

- Example
  - Character(3), Character Varying(20), Character(3), smallint are domains for **student** relation in XIT database.
  - These domains are basically "types" in SQL.
  - Each element in the tuple is ordered list of values drawn from respective domain.
  - Each position in the tuple-order represents an attribute.

“Relation” and “Table”



studid character(3)	name character varying(20)	progid character(3)	Class smallint
101	Rahul	BCS	2
102	Vikash	BEC	3
103	Shally	BEE	1
104	Alka	BEC	4
105	Ravi	BCS	1

- A relation can be viewed as a table(as depicted in figure above), where each row represent a tuple, and attribute values form columns.
- In RDBMs, it very common to refer attribute as column and tuple as row.

Relation in relational model

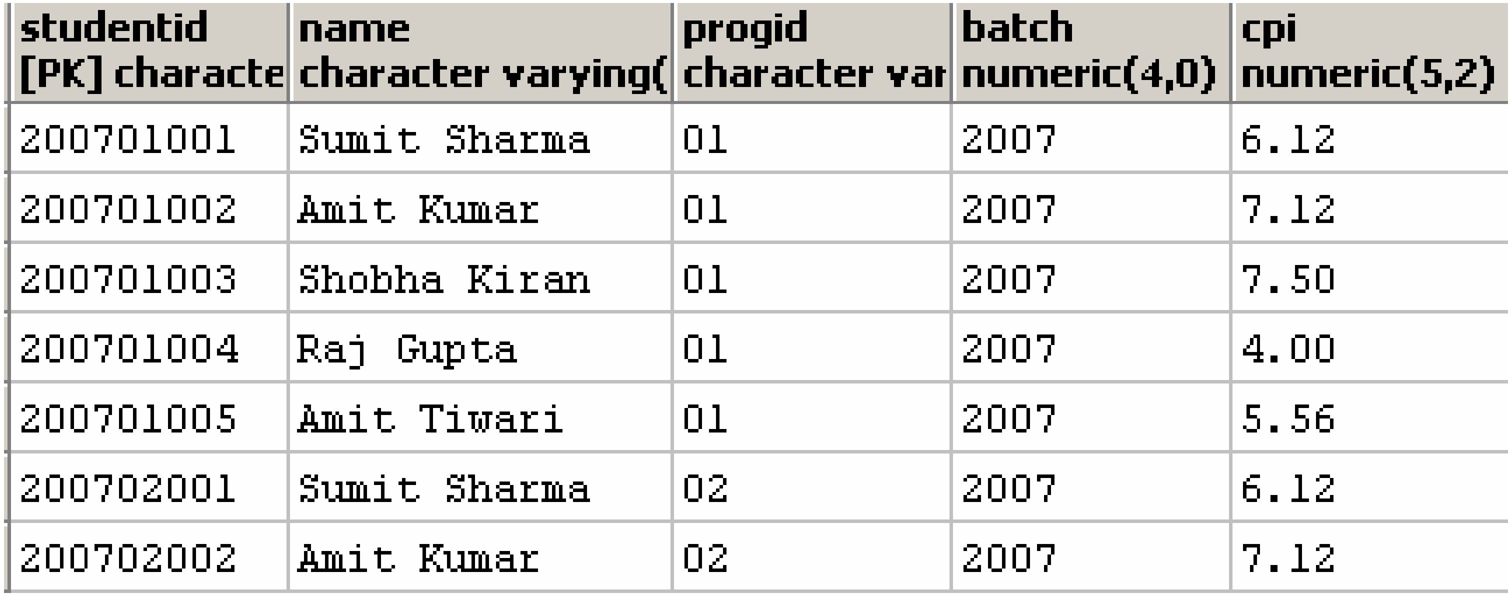
- Relation has two components-
  - Relation Schema, and
  - Relation itself (also referred as relation instance) – contains tuples, and represents relation state at a given time

Relation Schema

- Contains:
  - Name of relation
  - List of attributes and their domain, and
  - Constraints
    - Key
    - Integrity
    - Referential Integrity or so
- Example in SQL-DDL:
  Student(Studentid [int], name [varchar(20)], progid [smallint], batch [smallint], cpi [numeric(5,2]) with other details. Note in SQL, domains are defining through data-types.

Relation [relation instance]

- Example: **student** relation (instance)



studentid	name	progid	batch	cpi
[PK] characte	character varying(	character var	numeric(4,0)	numeric(5,2)
200701001	Sumit Sharma	01	2007	6.12
200701002	Amit Kumar	01	2007	7.12
200701003	Shobha Kiran	01	2007	7.50
200701004	Raj Gupta	01	2007	4.00
200701005	Amit Tiwari	01	2007	5.56
200702001	Sumit Sharma	02	2007	6.12
200702002	Amit Kumar	02	2007	7.12

Notations

- Primarily taken from Elmasri/navathe and let us use following 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)`** for relation [instance] of schema R

Tuples in a relation

- Tuple is represented by a row in tabular view

- Ordering of tuples in a relation r(R) are *not considered to be ordered*, even though they appear to be in the tabular form.
  - A relation (state) is defined as a set of tuples

Values in a tuple

- Tuple can be seen as ordered list of n-values; as
  `t = <v1, v2, ..., vn>`

- Tuple can also be seen as set of n (attrib-name, attrib-value) pairs.
  `t = {(A2,v2), (A1,v1), ..., (An,vn)}`

Values in a tuple

- Notationally, 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

Common Error

- Relational model is called relational because, it is collection of related tables – because there is relationship between tables

- Not True !

- It is relational because it is based on mathematical relations

Interpretation of Relational model

- A tuple represents some recordable facts - as a set of <attribute, value> pairs
- Schema of relation tells what all attributes a tuple can have values for
- A set of all such similar tuples (facts) is a relation

Degree and Cardinality of a relation

- Number of attributes is degree of a relation, while number of tuples is its cardinality

Integrity Constraints

- Generally, there are many restrictions or constraints on actual values in a database state.
- These constraints are derived from rules in the mini-real-world, which database represents.
- More formally, these are *database integrity constraints*, and ensure the database in *consistent state* over the period of time.
- Integrity constraints are specified as part of Database schema, and DBMS enforces these constraints on database states (operations leading to violation of any constraint is rejected by DBMS)

Integrity Constraints

- Following are main types of constraints-
  - Domain constraints
  - Key constraints
  - **NOT NULL** value constraints
  - Entity Integrity constraints
  - Referential Integrity Constraints
  - Other mini-word constraints – that may not be specified as part of Schema
- Remember that "Integrity constraints are properties of schema, and are to be enforced on every possible relation instance"

Domain Constraint

- Domain constraints specify that the value for each attribute A, must be an ***atomic*** value drawn from the domain of the attribute

- In SQL domains are specified by Data Types-
  - Integers- short, integer, long
  - Floating point numbers- float and double
  - Character String- Fixed and variable length strings
  - Others are- Date, Time, and Money
  - Subrange of above data types
  - Enumerated data types (explicitly listed values)

Key Constraints

- Key Constraints implies Distinct Tuples
- Every relation (state) is defined as a set of tuples, and therefore all tuples in a relation must be distinct.
- This means that no two tuple can have same combination of values for all of their attributes.
- Usually, there is some 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.

Key Constraints – Super Key

- Suppose a subset of attributes S, and no two tuples t1 and t2 in a relation state r of R, have same <attribute-value> pair set.
- In other we have the constraint that t1.S <> t2.S
- Such attribute set S is called **Super Key** of the relation schema R.
- A super key S specifies a uniqueness constraint that no two distinct tuple in a state r of R can have the same value for S.
- Every relation has at least one default super key- the set of all its attributes.

Key Constraints - Key

- Minimal Super key is Candidate Key or just **Key** of the relation
- **Key**: a super key can have redundant attributes, however. So there is more useful concept of Candidate key or key, which has no redundancy.
- A **Key** K of a relation schema R is super key of R with the additional property that removing any attribute A from K leaves a set of attributes K' that is not a super key of R.
- Note that a Key (or Super-Key) is said to be composite when involves more than one attribute.

Super-Key and Key

- Key and Super Key can be defined in each others terms
- Minimal Super-Key is Key.
- Any superset of Key is Super--Key.
- Example:
  - In the STUDENT relation, {StudID, Name, DateOfBirth} is a super key, or
  - Any set of attributes that includes StudID is a Super-Key. However {StudID, Name, DateOfBirth} is not a Key?

Key and (SQL) Primary Key

- In general a relation schema may have more than one key. For example- EnrollNo and StudentID in STUDENT relation.
- When the relation schema is defined on a RDBMS, SQL DDL is used, where you have notion of **Primary Key**, and it is specified when relation is created (using CREATE TABLE command).
- DBMS would typically (automatically) create an “index” on Primary key attributes.
- Any of the “Key” can be designated as **Primary Key**

NOT NULL value constraint

- NULL mean nothing. Can think of as attribute cell in a tuple “being empty”.

- A attribute can specified Not to be NULL (empty), then we call it NOT NULL value constraint.

- For example Name of Student may not be null.

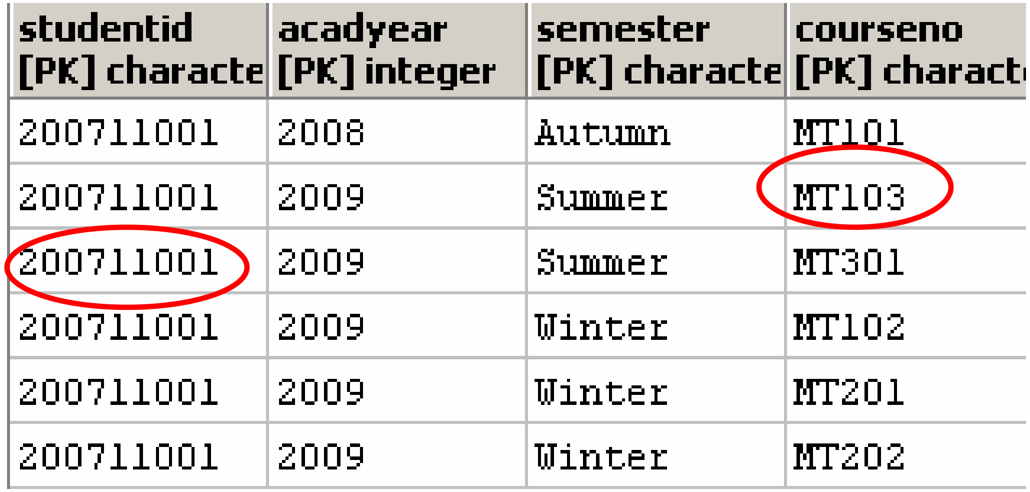
Entity integrity constraints

- In SQL when you define a **Primary Key**, a NOT NULL constraint is implicit - Primary key can not have NULL value
- Reasons for this- the primary key value is used to identify individual tuples in a relation. Allowing null value brings ambiguity
- Key constraints and entity integrity constraints are specified on individual relations.

Referential Integrity constraint

- Consider following relation, what if a user attempt to put ABCD as courseno, or 1234567 as student ID?

- What is wrong with such data, and how do we protect against such errors

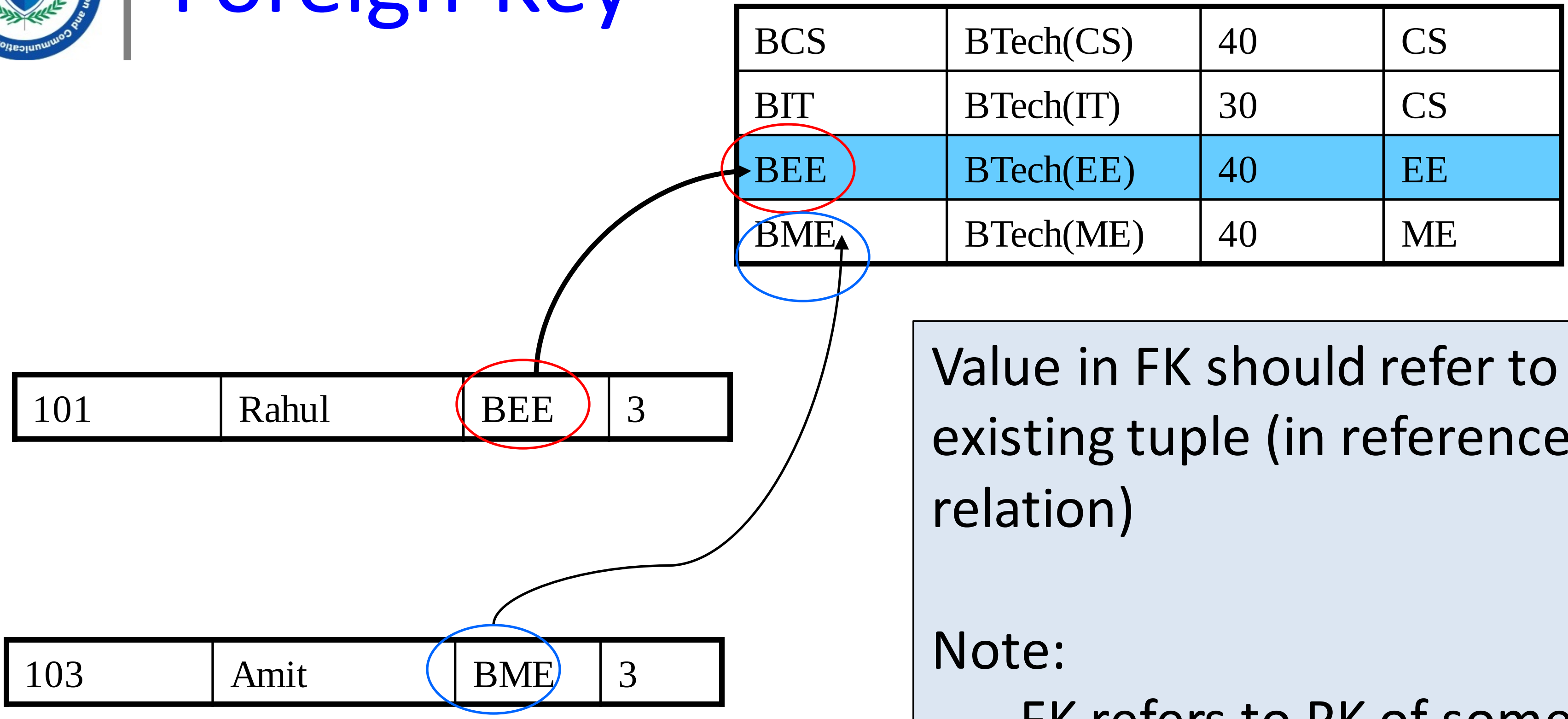


studentid [PK] characte	acadyear [PK] integer	semester [PK] characte	courseno [PK] charact
200711001	2008	Autumn	MT101
200711001	2009	Summer	MT103
200711001	2009	Summer	MT301
200711001	2009	Winter	MT102
200711001	2009	Winter	MT201
200711001	2009	Winter	MT202

Foreign Key

- Attributes Student ID and Course ID are attributes used for referring another entities, a student, and a course respectively.
- These attributes are basically here in this relation representing some other entity are called “Foreign Keys”
- Foreign key is basically a mechanism of referring existing entitles, that is referring a tuple in some other relation (and may be in same relation?).

Foreign Key



BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

101	Rahul	BEE	3

103	Amit	BME	3

Value in FK should refer to existing tuple (in referenced relation)

Note:

- FK refers to PK of some other relation
- Domain of FK is same as of corresponding PK

Foreign Key - Rules

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

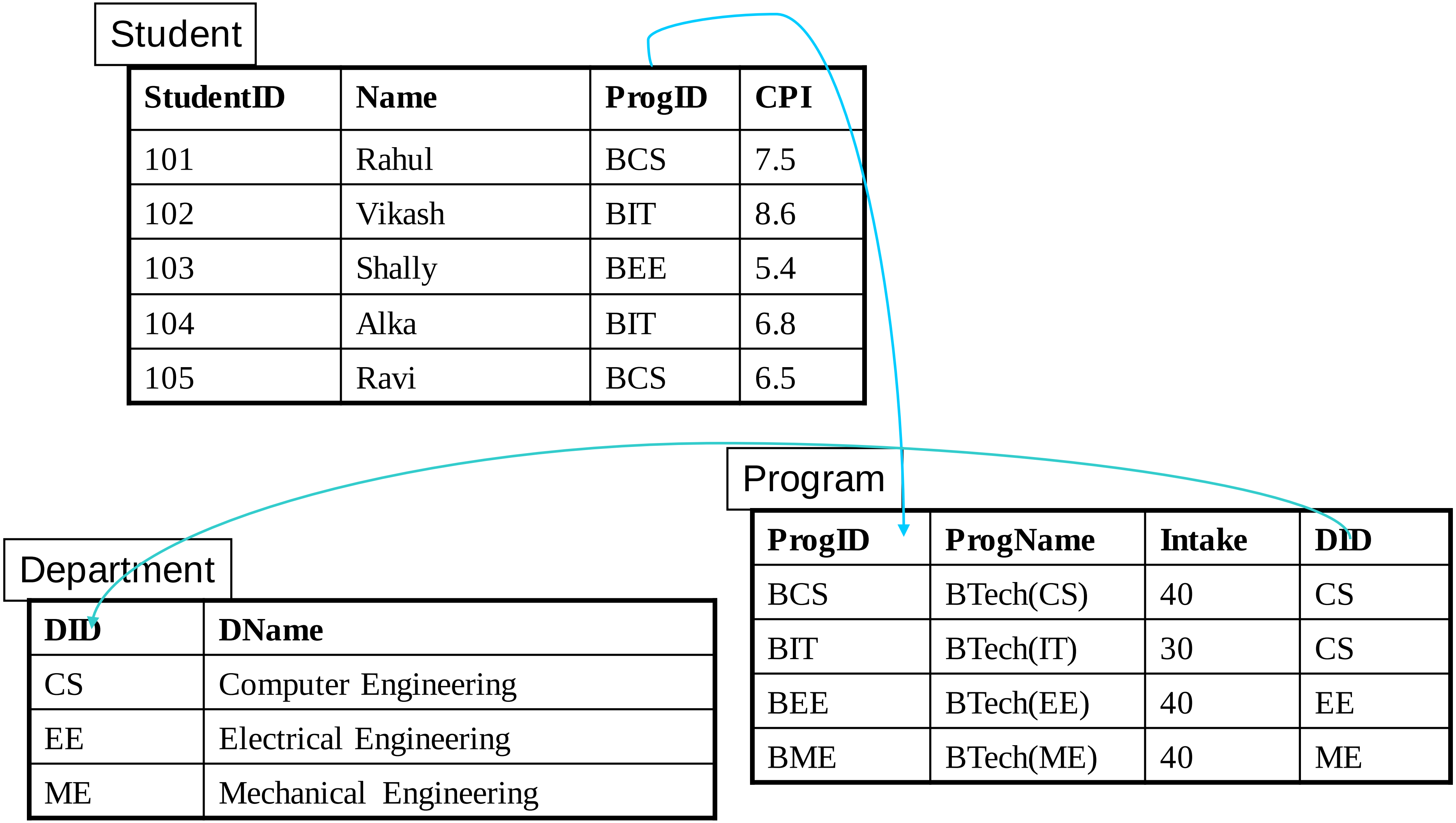
1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2.
2. A value of FK in a tuple t1 of the current state r1(R) either occurs as a value of PK (or any candidate key) for some tuple t2 in the current state r2(R) or is null.
   In the former case, we have t1.FK = t2.PK, and we say that the tuple t1 refers to the tuple t2.

Referential Integrity constraint

- Referential Integrity constraint are specified between two relation (schemas), and is used to maintain the consistency among tuples of two relations.

- Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Observe Referential Integrities here…



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

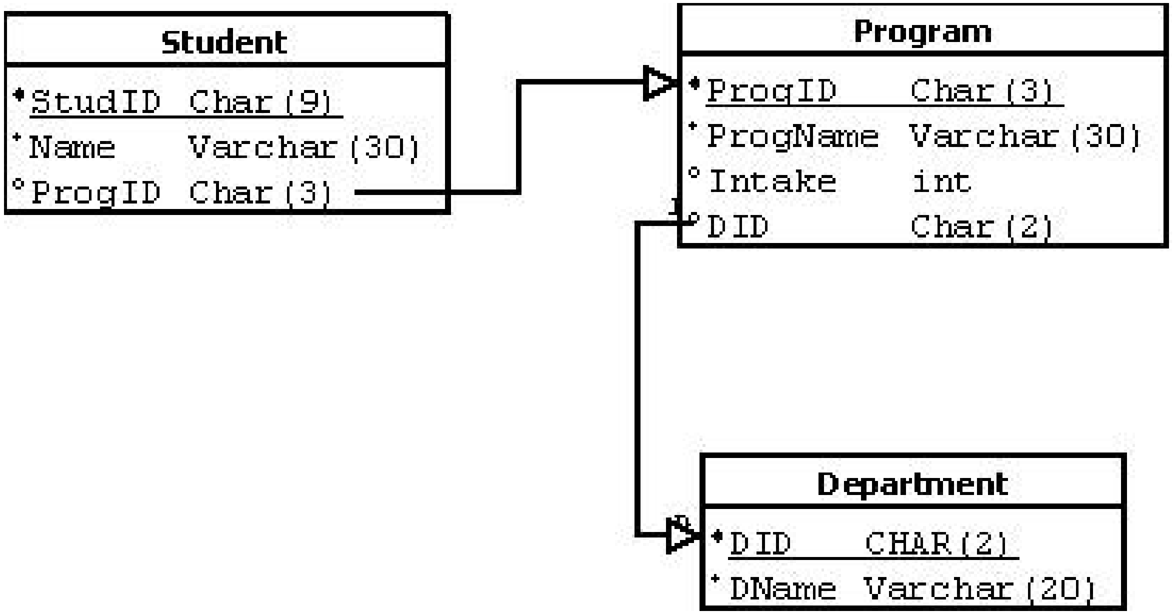
Program

ProgID	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

Can you Identify
Constraints in XIT schema?

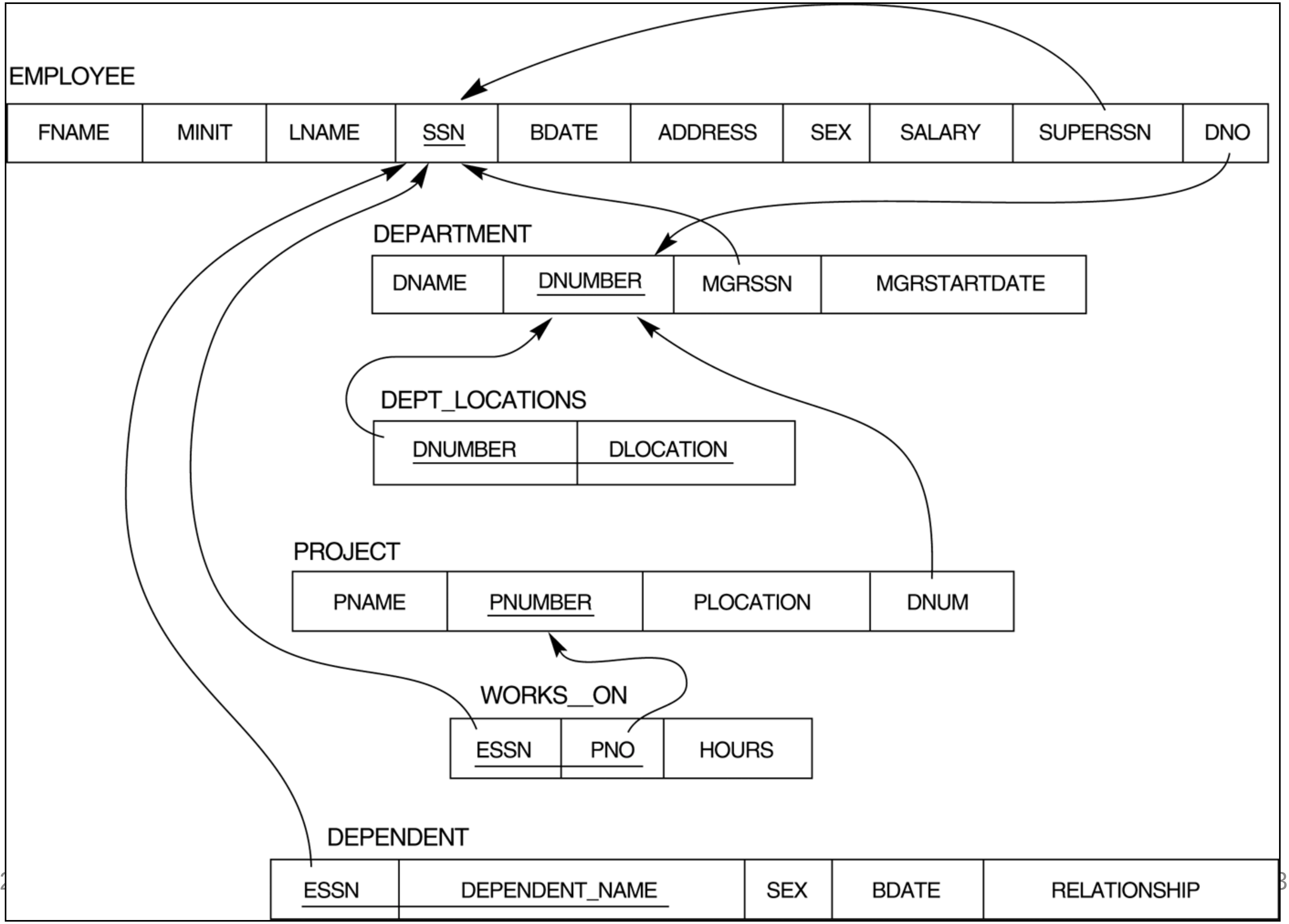


Student	
◆ StudID	Char(9)
* Name	Varchar(30)
° ProgID	Char(3)

Program	
◆ ProgID	Char(3)
* ProgName	Varchar(30)
° Intake	int
° DID	Char(2)

Department	
◆ DID	CHAR(2)
* DName	Varchar(20)

Understand Company Schema



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS__ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Other Types of Constraints

- Semantic Integrity Constraints
  - based on application semantics and cannot be expressed by the model
  - For example
    - Salary of person can not be more than his/her supervisor
    - “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
  - Such type of constraints either can be enforced by application programs or by a general purpose “constraint specification language”
  - SQL-99 allows Triggers and Assertions to allow for some of these