mood-book

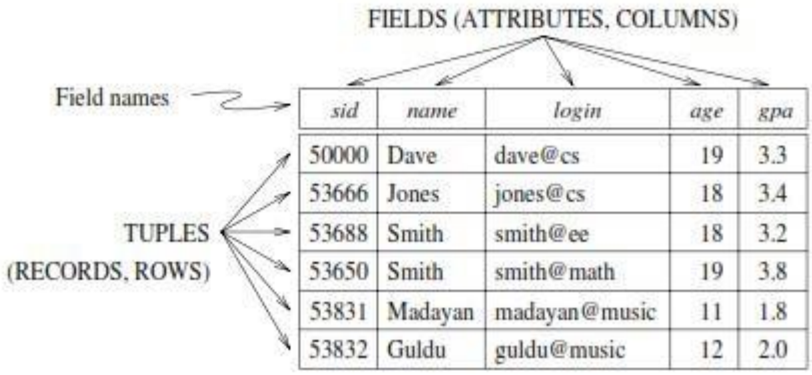
UNIT-2

1. Introduction to Relational Model

The main construct for representing data in the relational model is a relation.Arelation consists of a **“Relation Schema”** and a **“Relation Instance”**.

- The Relation Schema the relation’s name, the name of each field and the domain of each field. A domain is referred to in a relation schema by the domain name and has set of associated values.
  Ex: Students(sidchar(10), name char(10), loginchar(10), age: integer, gpa: real)
  This says, for instance, that the field named sid has a domain named char.
- The Relation Instances of a relation. An instance of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema. A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields.

  Ex: An Instance S1 of the Students Relation



A relation schema specifies the domain of each field or column in the relation instance.These domain constraints in the schema specify an important condition that wewant each instance of the relation to satisfy: The values that appear in a column mustbe drawn from the domain associated with that column. Thus, the domain of a fieldis essentially the type of that field, in programming language terms, and restricts thevalues that can appear in the field.

More formally, let $R(f_1{:}D_1, :::, f_n{:}D_n)$ be a relation schema, and for each f_i,1<=i<=n,let Dom_i be the set of values associated with the domain named D_i. An instance of Rthat satisfies the domain constraints in the schema is a set of tuples with n fields:

$$\{ \langle f_1 : d_1, \ \ldots \ , f_n : d_n \rangle \ | \ d_1 \in Dom_1, \ \ldots \ , d_n \in Dom_n \}$$

The angular brackets <……> identify the fields of a tuple. Using this notation, the firstStudents tuple shown in above Figure is written as <sid: 50000, name: Dave,

login:dave@cs, age: 19, gpa: 3.3>. The curly brackets {:::} denote a set. The vertical bar | should be read `such that,' the symbol € should be read`in,' and the expression to the right of the vertical bar is a condition that must besatisfied by the field values of each tuple in the set. Thus, an instance of R is definedas a set of tuples. The fields of each tuple must correspond to the fields in the relationschema.

Therefore, relation instance meansrelation instance that satisfies the domain constraints in the relation schema.

Degree/Arity of a Relation: The number of fields.

Cardinality of Relation Instance: The number of tuples in it

Ex: the degree of therelation (the number of columns) is five, and the cardinality of this instance is six.

A relational database is a collection of relations with distinct relation names. Therelational database schema is the collection of schemas for the relations in thedatabase.

Creating and Modifying Relations Using SQL

The subset of SQL that supportsthe creation, deletion, and modification of tables is called the Data Definition Language(DDL).

The CREATE TABLE statement is used to define a new table.To create the Studentsrelation, we can use the following statement:

CREATE TABLE Students (sid CHAR(20),

name CHAR(30),

login CHAR(20),

age INTEGER,

gpa REAL);

Tuples are inserted using the INSERT command. We can insert a single tuple into theStudents table as follows:

INSERT Students values (53688, `Smith', `smith@ee', 18, 3.2);

We can delete tuples using the DELETE command. We can delete all Students tupleswith name equal to Smith using the command:

DELETE

FROM Students S

WHERE S.name = `Smith';

We can modify the column values in an existing row using the UPDATE command.

Ex: we can increment the age and decrement the gpa of the student with sid53688:

UPDATE Students S

SET S.age = S.age + 1, S.gpa = S.gpa - 1

WHERE S.sid = 53688;

2. Integrity Constraints over Relations

An integrity constraint (IC) is acondition that is specified on a database schema, and restricts the data that can bestored in an instance of the database. If a database instance satisfies all the integrityconstraints specified on the database schema, it is a legal instance. A DBMS enforcesintegrity constraints, in that it permits only legal instances to be stored in the database.

Integrity constraints are specified and enforced at different times:

1. When the DBA or end user defines a database schema, he or she specifies the ICsthat must hold on any instance of this database.

2. When a database application is run, the DBMS checks for violations and disallowschanges to the data that violate the specified ICs.

Key Constraints

A key constraint is a statementthat a certain minimal subset of the fields of a relation is a unique identifier for a tuple.A set of fields that uniquely identifies a tuple according to a key constraint is calleda candidate key for the relation.

A relation may have several candidate keys. For example, the login and age fields ofthe Students relation may, taken together, also identify students uniquely. That is {login, age} is also a key. It may seem that login is a key, since no two rows in theexample instance have the same login value. However, the key must identify tuplesuniquely in all possible legal instances of the relation. By stating that {login, age} isa key, the user is declaring that two students may have the same login or age, but notboth.

Out of all the available candidate keys, a database designer can identify a primarykey. Intuitively, a tuple can be referred to from elsewhere in the database by storingthe values of its primary key fields.

Ex: we can refer to a Students tuple by storing its sid value.

Specifying Key Constraints in SQL

In SQL we can declare that a subset of the columns of a table constitute a key byusing the UNIQUE constraint. At most one of these `candidate' keys can be declaredto be a primary key, using the PRIMARY KEY constraint.

Let us revisit our example table definition and specify key information:

CREATE TABLE Students (sid CHAR(20),

name CHAR(30),

login CHAR(20),

age INTEGER,

gpa REAL,

PRIMARY KEY (sid));

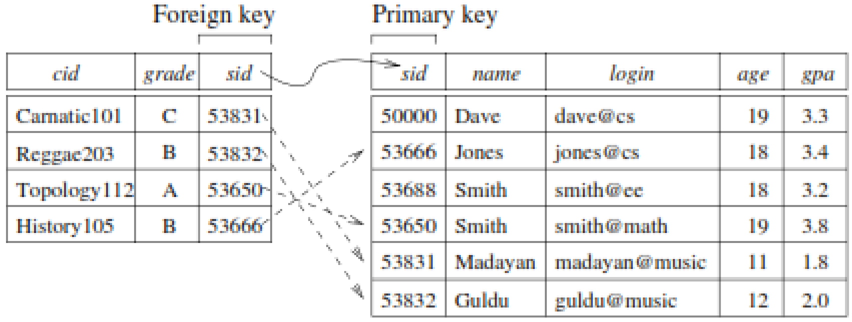
Foreign Key Constraints

Sometimes the information stored in a relation is linked to the information stored inanother relation. If one of the relations is modified, the other must be checked, andperhaps modified, to keep the data consistent. An IC involving both relations mustbe specified if a DBMS is to make such checks. The most common IC involving tworelations is a foreign key constraint.Suppose that in addition to Students, we have a second relation:

Enrolled(sid: char(10), cid: char(10), grade: char(10));

To ensure that only bonafide students can enrol in courses, any value that appears inthe sidfield of an instance of the Enrolled relation should also appear in the sid fieldof some tuplein the Students relation. The sid field of Enrolled is called a foreignkey and refers to Students.The foreign key in the referencing relation (Enrolled, inour example) must match the primary key of the referenced relation (Students), i.e.,it must have the same number of columns and compatible data types, although thecolumn names can be different.

This constraint is illustrated in the following Figure shows, there may well besome students who are not referenced from Enrolled (e.g., the student with sid=50000).However, every sid value that appears in the instance of the Enrolled table appears inthe primary key column of a row in the Students table.



Enrolled (Referencing relation) Students (Referenced relation)

If we try to insert the tuple <55555, Art104, A> into E1, the IC is violated becausethere is no tuple in S1 with the id 55555; the database system should reject suchan insertion. Similarly, if we delete the tuple <53666, Jones, jones@cs, 18, 3.4> fromS1, we violate the foreign key constraint because the tuple <53666, History105, B>in E1 contains sid value 53666, the sid of the deleted Students tuple. The DBMSshould disallow the deletion or, perhaps, also delete the Enrolled tuple that refers tothe deleted Students tuple.

Specifying Foreign Key Constraints in SQL

Let us define Enrolled(sid: string, cid: string, grade: string):

CREATE TABLE Enrolled (sid CHAR(20),

cid CHAR(20),

grade CHAR(10),

PRIMARY KEY (sid, cid),

FOREIGN KEY (sid) REFERENCES Students)

The foreign key constraint states that every sid value in Enrolled must also appear inStudents, that is, sid in Enrolled is a foreign key referencing Students. Incidentally,the primary key constraint states that a student has exactly one grade for each coursethat he or she is enrolled in. If we want to record more than one grade per studentper course, we should change the primary key constraint.

General Constraints

For example, we may require that student ages be within a certain range of values;given such an IC specification, the DBMS will reject inserts and updates that violatethe constraint. This is very useful in preventing data entry errors. If we specify thatall students must be at least 16 years old, the instance of Students shown in above Figure(studenttable) illegal because two students are underage. If we disallow the insertion of thesetwo tuples, we have a legal instance, as shown in following fig.,

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8

3. Enforcing Integrity Constraints

ICs are specified when a relation is created and enforced whena relation is modified. The impact of domain, PRIMARY KEY,andUNIQUE constraintsis straightforward: if an insert, delete, or update command causes a violation, it isrejected. Potential IC violation is generally checked at the end of each SQL statementexecution, although it can be deferred until the end of the transaction executing the statement.

Consider the instance S1 of Students shown in Figure (student table) . The following insertionviolates the primary key constraint because there is already a tuple with the sid 53688,and it will be rejected by the DBMS:

INSERT

INTO Students (sid, name, login, age, gpa)

VALUES (53688, `Mike', `mike@ee', 17, 3.4);

The following insertionviolates the constraint that the primary key cannot containnull:

INSERT

INTO Students (sid, name, login, age, gpa)

VALUES (null, `Mike', `mike@ee', 17, 3.4);

A similar problem arises whenever we try to insert a tuple with a value ina field that is not in the domain associated with that field, i.e., whenever we violatea domain constraint. Deletion does not cause a violation of domain, primary key orunique constraints. However, an update can cause violations, similar to an insertion:

UPDATE Students S

SET S.sid = 50000

WHERE S.sid = 53688;

This update violates the primary key constraint because there is already a tuple withsid 50000.

Cascading Referential Integrity Constraints

The REFERENCES clauses of the CREATE TABLE and ALTER TABLE statements support the ON DELETE and ON UPDATE clauses. Cascading actions can also be definedby using the Foreign Key Relationships dialog box:

- [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]
- [ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]

NO ACTION is the default if ON DELETE or ON UPDATE is not specified.

ON DELETE NO ACTION

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE statement is rolled back.

ON UPDATE NO ACTION

Specifies that if an attempt is made to update a key value in a row whose key is referenced by foreign keys in existing rows in other tables, an error is raised and the UPDATE statement is rolled back.

CASCADE, SET NULL and SET DEFAULT allow for deletions or updates of key values to affect the tables defined to have foreign key relationships that can be traced back to the table on which the modification is performed. If cascading referential actions have also been defined on the target tables, the specified cascading actions also apply for those rows deleted or updated. CASCADE cannot be specified for any foreign keys or primary keys that have a timestamp column.

ON DELETE CASCADE

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows that contain those foreign keys are also deleted.

ON UPDATE CASCADE

Specifies that if an attempt is made to update a key value in a row, where the key value is referenced by foreign keys in existing rows in other tables, all the values that make up the foreign key are also updated to the new value specified for the key.

ON DELETE SET NULL

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all the values that make up the foreign key in the rows that are referenced are set to NULL. All foreign key columns of the target table must be nullable for this constraint to execute.

ON UPDATE SET NULL

Specifies that if an attempt is made to update a row with a key referenced by foreign keys in existing rows in other tables, all the values that make up the foreign key in the rows that are referenced are set to NULL. All foreign key columns of the target table must be nullable for this constraint to execute.

ON DELETE SET DEFAULT

Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all the values that make up the foreign key in the rows that are referenced are set to their default value. All foreign key columns of the target table must have a default definition for this constraint to execute. If a column is nullable, and there is no explicit default value set, NULL becomes the implicit default value of the column. Any not null values that are set because of ON DELETE SET DEFAULT must have corresponding values in the primary table to maintain the validity of the foreign key constraint.

ON UPDATE SET DEFAULT

Specifies that if an attempt is made to update a row with a key referenced by foreign keys in existing rows in other tables, all the values that make up the foreign key in the rows that are referenced are set to their default value. All foreign key columns of the target table must have a default definition for this constraint to execute. If a column is nullable, and there is no explicit default value set, NULL becomes the implicit default value of the column. Any non-null values that are set because of ON UPDATE SET DEFAULT must have corresponding values in the primary table to maintain the validity of the foreign key constraint.

Ex: creating tables named `parent` and `child`, such that the `child` table contains a foreign key that references the `par_id` column in the `parent` table:

```
CREATE TABLE parent(par_id  INT NOT NULL,
                    PRIMARY KEY (par_id));

CREATE TABLE child(par_id  INT NOT NULL,
                   child_id INT NOT NULL,
                    PRIMARY KEY (par_id, child_id),
                    FOREIGN KEY (par_id) REFERENCES parent (par_id)
                     ON DELETE CASCADE
                     ON UPDATE CASCADE);
```

The foreign key in this case uses ON DELETE CASCADE to specify that when a record is deleted from the parent table, MySQL also should remove child records with a matching par_id value automatically. ON UPDATE CASCADE indicates that if a parent record par_id value is changed, MySQL also should change any matching par_id values in the child table tothe new value.

4. Querying Relational Data

A relational database query (query, for short) is a question about the data, and theanswer consists of a new relation containing the result.

Ex: if we might wantto find all students younger than 18 or all students enrolled in Reggae203.

A querylanguage is a specialized language for writing queries.SQL is the most popular commercial query language for a relational DBMS. We nowpresent some SQL examples that illustrate how easily relations can be queried. Considerthe instance of the Students relation shown in Figure (student table). We can retrieve rowscorresponding to students who are younger than 18 with the following SQL query:

```
SELECT *
FROM Students S
WHERE S.age< 18;
```

The symbol * means that we retain all fields of selected tuples in the result. Tounderstand this query, think of S as a variable that takes on the value of each tuplein Students, one tuple after the other. The condition S.age< 18 in the WHERE clausespecifies that we want to select only tuples in which the age field has a value less than18. This query evaluates to the relation shown in Figure

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

In addition to selecting a subset of tuples, a query can extract a subset of the fieldsof each selected tuple. We can compute the names and logins of students who areyounger than 18 with the following query:

SELECT S.name, S.login

FROM Students S

WHERE S.age< 18;

The following Figure shows the answer to this query; it is obtained by applying the selectionto the instance S1 of Students, followed byremoving unwanted fields of student tablespecified earlier

name	*login*
Madayan	madayan@music
Guldu	guldu@music

5. Logical Database Design: ER to Relational

The ER model is convenient for representing an initial, high-level database design.Given an ER diagram describing a database, there is a standard approach to generatinga relational database schema that closely approximates the ER design.

Entity Sets to Tables

An entity set is mapped to a relation in a straightforward way: Each attribute of theentity set becomes an attribute of the table. Note that we know both the domain ofeach attribute and the (primary) key of an entity set.

Ex: Consider the Employees entity set with attributes ssn, name, and lot shown in Figure below.

A possible instance of the Employees entity set, containing three Employeesentities, is shown in Figure 3.9 in a tabular format.

ssn	*name*	*lot*
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

The following SQL statement captures the preceding information, including the domainconstraints and key information:

CREATE TABLE Employees (ssn CHAR(11),

name CHAR(30),

lot INTEGER,

PRIMARY KEY (ssn));

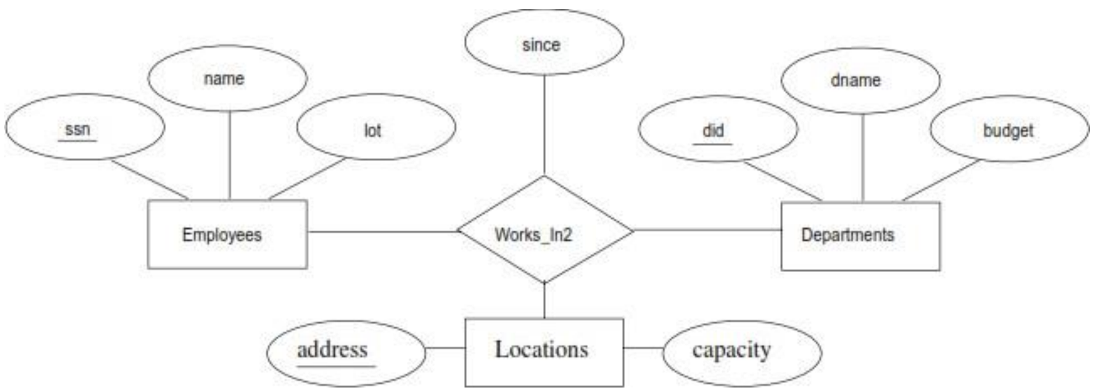
Relationship Sets (without Constraints) to Tables

A relationship set, like an entity set, is mapped to a relation in the relational model.We begin by considering relationship sets without key and participation constraints,and we discuss how to handle such constraints in subsequent sections. To representa relationship, we must be able to identify each participating entity and give valuesto the descriptive attributes of the relationship. Thus, the attributes of the relationinclude:

- The primary key attributes of each participating entity set, as foreign key fields.
- The descriptive attributes of the relationship set.

The set of nondescriptive attributes is a superkey for the relation. If there are no keyconstraints, this set of attributes is a candidate key.

Ex: Consider the Works_In2 relationship set shown in below Figure. Each department hasoffices in several locations and we want to record the locations at which each employee works



All the available information about the Works In2 table is captured by the followingSQLdefinition:

CREATE TABLE Works_In2 (ssn CHAR(11),

did INTEGER,

address CHAR(20),

since DATE,

PRIMARY KEY (ssn, did, address),

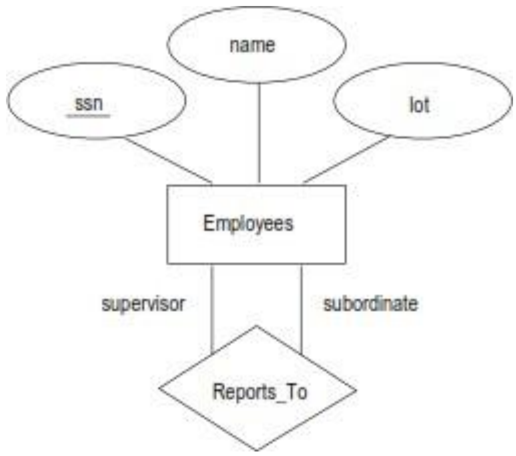
FOREIGN KEY (ssn) REFERENCES Employees,

FOREIGN KEY (address) REFERENCES Locations,

FOREIGN KEY (did) REFERENCES Departments);

Note that the address, did, and ssnfields cannot take on null values. Because these fields are part of the primary key for Works_In2, a NOT NULL constraint is implicitfor each of these fields. This constraint ensures that these fields uniquely identifya department, an employee, and a location in each tuple of Works_In.

Ex: consider the ReportsTo relationship set shown in below Figure. The role indicators supervisor and subordinate are used to create meaningful field names.



CREATE statement for the Reports_To table:

CREATE TABLE Reports_To (

supervisor_ssn CHAR(11),

Subordinate_ssnCHAR(11),

PRIMARY KEY (supervisor_ssn, subordinate_ssn),

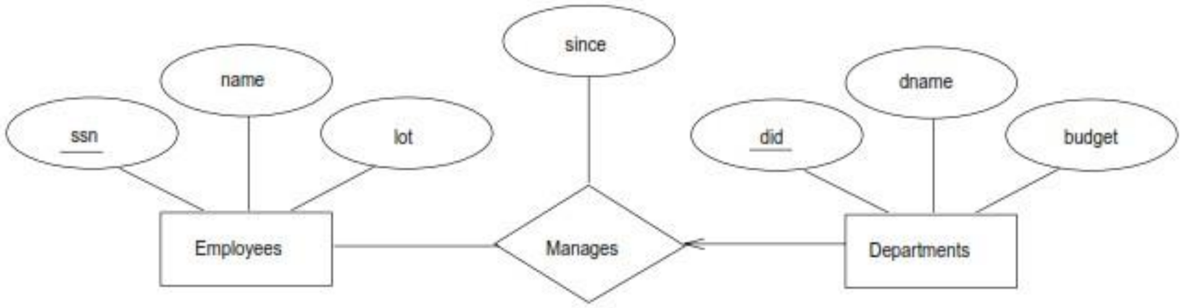
FOREIGN KEY (supervisor_ssn) REFERENCES Employees(ssn), FOREIGN KEY (subordinate_ssn) REFERENCES Employees(ssn));

Observe that we need to explicitly name the referenced field of Employees because the field name differs from the name(s) of the referring field(s).

Translating Relationship Sets with Key Constraints

If a relationship set involves n entity sets and some m of them are linked via arrowsin the ER diagram, the key for any one of these m entity sets constitutes a key forthe relation to which the relationship set is mapped. Thus we have m candidate keys,and one of these should be designated as the primary key.

Consider the relationship set Manages shown in below Figure.



The table correspondingto Manages has the attributes ssn,did,since. However, because each department hasat most one manager, no two tuples can have the same did value but differ on the ssnvalue. A consequence of this observation is that did is itself a key for Manages; indeed,the set did, ssn is not a key (because it is not minimal). The Manages relation can bedefined using the following SQL statement:

CREATE TABLE Manages (ssn CHAR(11),

did INTEGER,

since DATE,

PRIMARY KEY (did),

FOREIGN KEY (ssn) REFERENCES Employees,

FOREIGN KEY (did) REFERENCES Departments);

A second approach to translating a relationship set with key constraints is often superior because it avoids creating a distinct table for the relationship set. The ideais to include the information about the relationship set in the table corresponding tothe entity set with the key, taking advantage of the key constraint. In the Managesexample, because a department has at most one manager, we can add the key fields ofthe Employees tuple denoting the manager and the since attribute to the Departmentstuple.

This approach eliminates the need for a separate Manages relation, and queries asking for a department's manager can be answered without combining information from tworelations. The only drawback to this approach is that space could be wasted if severaldepartments have no managers. In this case the added fields would have to be filled with null values. The first translation (using a separate table for Manages) avoids thisinefficiency, but some important queries require us to combine information from tworelations, which can be a slow operation.

The following SQL statement, defining a Dept_Mgr relation that captures the informationin both Departments and Manages, illustrates the second approach to translating relationship sets with key constraints:

CREATE TABLE DeptMgr(did INTEGER,

dname CHAR(20),

budget REAL,

ssn CHAR(11),

since DATE,

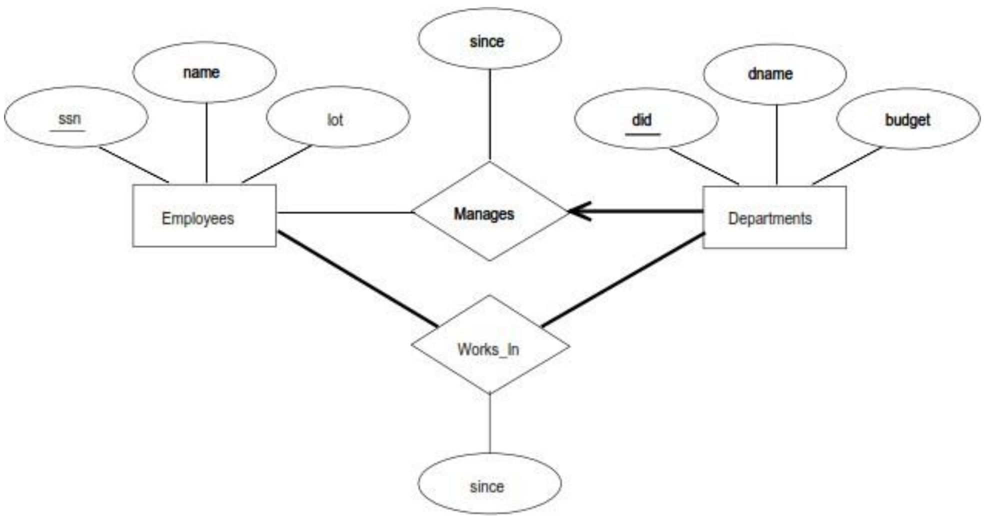
PRIMARY KEY (did),

FOREIGN KEY (ssn) REFERENCES Employees);

Note that ssn can take on null values.This idea can be extended to deal with relationship sets involving more than two entitysets.

Translating Relationship Sets with Participation Constraints

Consider the ER diagram in below Figure, which shows two relationship sets, Managesand Works_In.



Every department is required to have a manager, due to the participationconstraint,and at most one manager, due to the key constraint. The following SQL statementreflects the second translation approach, and uses the keyconstraint:

CREATE TABLE Dept_Mgr(did INTEGER,

dname CHAR(20),

budget REAL,

ssn CHAR(11) NOT NULL,

since DATE,

PRIMARY KEY (did),

FOREIGN KEY (ssn) REFERENCES Employees

ON DELETE NO ACTION);

It also captures the participation constraint that every department must have a manager: Because ssn cannot take on null values, each tuple of Dept_Mgr identifies a tuplein Employees (who is the manager). The NO ACTION specification, which is the defaultand need not be explicitly specified, ensures that an Employees tuple cannot be deletedwhile it is pointed to by a Dept_Mgr tuple. If we wish to delete such an Employeestuple, we must first change the Dept_Mgr tuple to have a new employee as manager.

Unfortunately, there are many participation constraints that we cannot capture using SQL, short of using table constraints or assertions. Table constraints and assertionscan be specified using the full power of the SQL query language and are very expressive, but also very expensive to check and enforce.

Ex: we cannot enforce the participation constraints on the Works_In relation without using these general constraints. To see why, consider the Works_In relation obtained by translating the ER diagram into relations. It contains field sssn and did, which are foreign keys referring to Employees and Departments. To ensure totalparticipation of Departments in Works_In, we have to guarantee that every did value inDepartments appears in a tuple of Works_In. We could try to guarantee this conditionby declaring that did in Departments is a foreign key referring to Works_In, but thisis not a valid foreign key constraint because did is not a candidate key for Works_In.

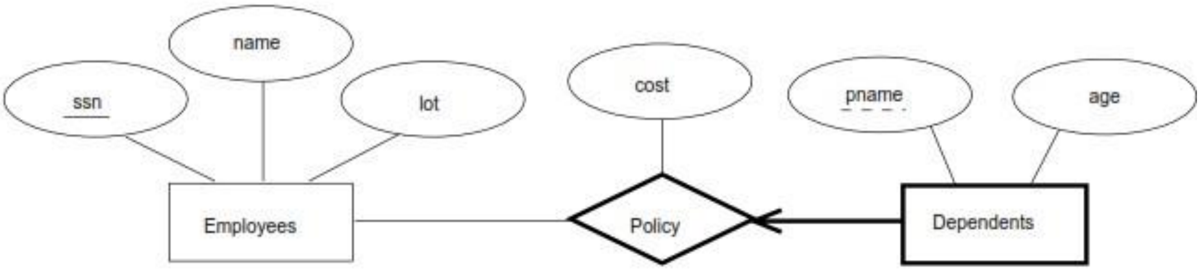
To ensure total participation of Departments in Works_In using SQL, we need anassertion. We have to guarantee that every did value in Departments appears in atuple of Works_In; further, this tuple of Works_In must also have non null values inthe fields that are foreign keys referencing other entity sets involved in the relationship(in this example, the ssn field). We can ensure the second part of this constraint byimposing the stronger requirement that ssn in Works_In cannot contain null values.

In fact, the Manages relationship set exemplifies most of the participation constraintsthat we can capture using key and foreign key constraints. Manages is a binary relationshipset in which exactly one of the entity sets (Departments) has a key constraint,and the total participation constraint is expressed on that entity set.

Translating Weak Entity Sets

A weak entity set always participates in a one-to-many binary relationship and has akey constraint and total participation. The second translation approach ideal in this case, but we must take into account the fact that the weakentity has only a partial key. Also, when an owner entity is deleted, we want all ownedweak entities to be deleted.

Ex: Consider the Dependents weak entity set shown in below Figure, with partial keypname.A Dependents entity can be identified uniquely only if we take the key of the owningEmployees entity and the pname of the Dependents entity, and the Dependents entitymust be deleted if the owning Employees entity is deleted.



We can capture the desired semantics with the following definition of theDep_Policyrelation:

CREATE TABLE Dep_Policy(pname CHAR(20),

age INTEGER,

cost REAL,

ssn CHAR(11),

PRIMARY KEY (pname, ssn),

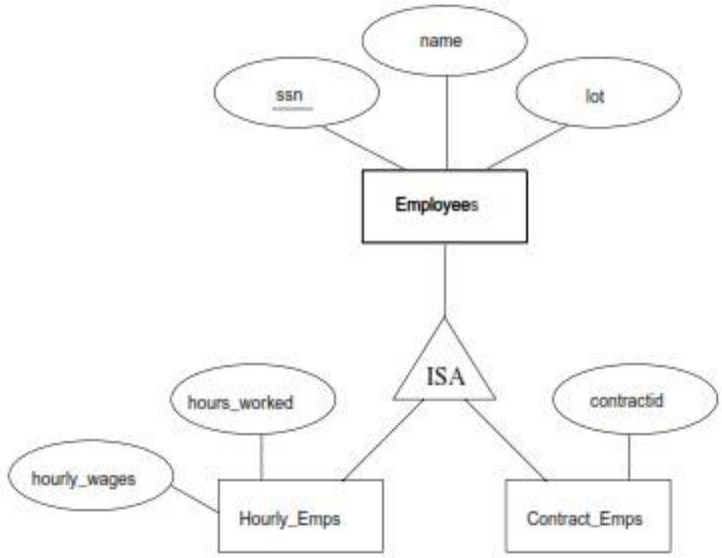
FOREIGN KEY (ssn) REFERENCES Employees

ON DELETE CASCADE);

Observe that the primary key is <pname, ssn>, since Dependents is a weak entity. Thisconstraint is a change with respect to the translation. Wehave to ensure that every Dependents entity is associated with an Employees entity(the owner), as per the total participation constraint on Dependents. That is, ssncannot be null. This is ensured because ssn is part of the primary key. The CASCADEoption ensures that information about an employee's policy and dependents is deletedif the corresponding Employees tuple is deleted.

Translating Class Hierarchies

We present the two basic approaches to handling ISA hierarchies by applying them to the ER diagram shown in below Figure:



1. We can map each of the entity sets Employees, Hourly_Emps, and Contract_Empsto a distinct relation. The Employees relation is created previously. Wediscuss Hourly_Emps here; Contract_Emps is handled similarly. The relation forHourly_Emps includes the hourly wages and hours worked attributes of Hourly_Emps.It also contains the key attributes of the superclass (ssn, in this example), whichserve as the primary key for HourlyEmps, as well as aforeign key referencingthe superclass (Employees). For each Hourly_Emps entity, the values of the name and lot attributes are stored in the corresponding row of the superclass (Employees).Note that if the superclass tuple is deleted, the delete must be cascaded toHourly_Emps.

2. Alternatively, we can create just two relations, corresponding to Hourly_Empsand Contract_Emps. The relation for Hourly_Emps includes all the attributesof Hourly_Emps as well as all the attributes of Employees (i.e., ssn, name, lot,hourly_wages, hours worked).

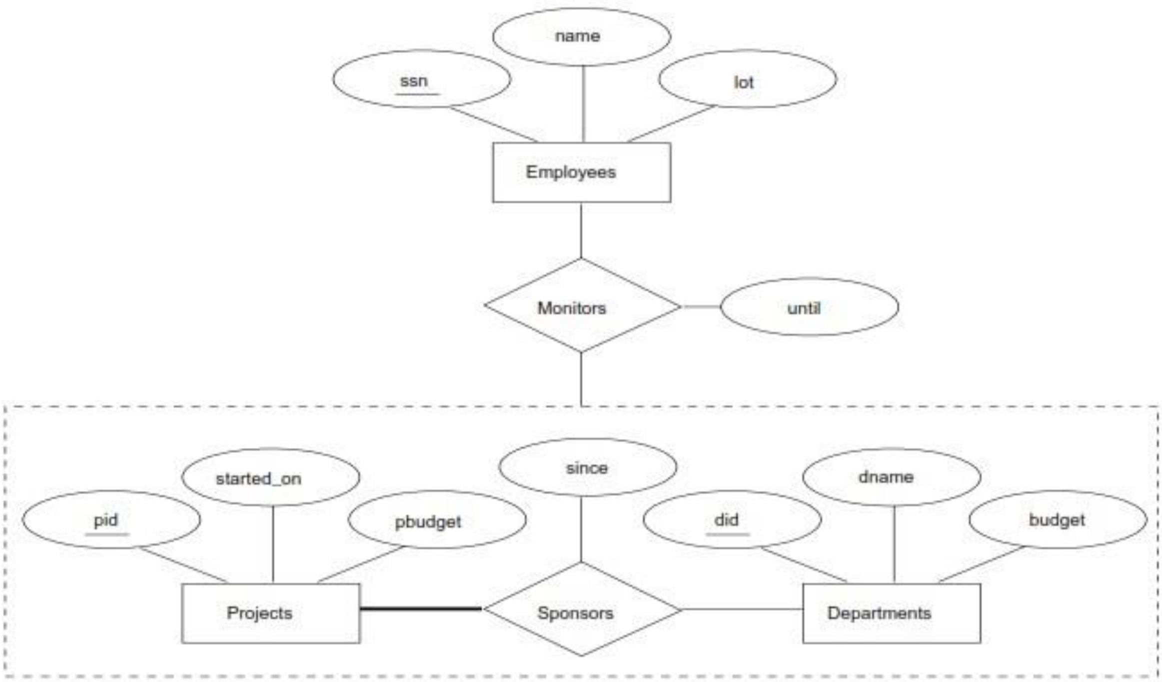
The first approach is general and is always applicable. Queries in which we want toexamine all employees and do not care about the attributes specific to the subclassesare handled easily using the Employees relation. However, queries in which we wantto examine, say, hourly employees, may require us to combine Hourly_Emps (or Contract_Emps,as the case may be) with Employees to retrieve name and lot.

The second approach is not applicable if we have employees who are neither hourlyemployees nor contract employees, since there is no way to store such employees. Also,if an employee is both anHourly_Emps and a Contract_Emps entity, then the nameand lot values are stored twice. This duplication can lead to some of the anomalies. A query that needs to examine all employees must nowexamine two relations. On the other hand, a query that needs to examine only hourly_employees can now do so by examining just one relation. The choice between theseapproaches clearly depends on the semantics of the data and the frequency of commonoperations.

In general, overlap and covering constraints can be expressed in SQL only by usingassertions.

Translating ER Diagrams with Aggregation

Translating aggregation into the relational model is easy because there is no real distinctionbetween entities and relationships in the relational model.Consider the ER diagram shown below Figure.



The Employees, Projects, and Departments entity sets and the Sponsors relationship set are mapped. For the Monitors relationship set, we create a relation with thefollowing attributes: the key attributes of Employees (ssn), the key attributes of Sponsors (did, pid), and the descriptive attributes of Monitors (until). This translation isessentially the standard mapping for a relationship set.

There is a special case in which this translation can be refined further by droppingtheSponsors relation. Consider the Sponsors relation. It has attributes pid, did, andsince, and in general we need it (in addition to Monitors) for two reasons:

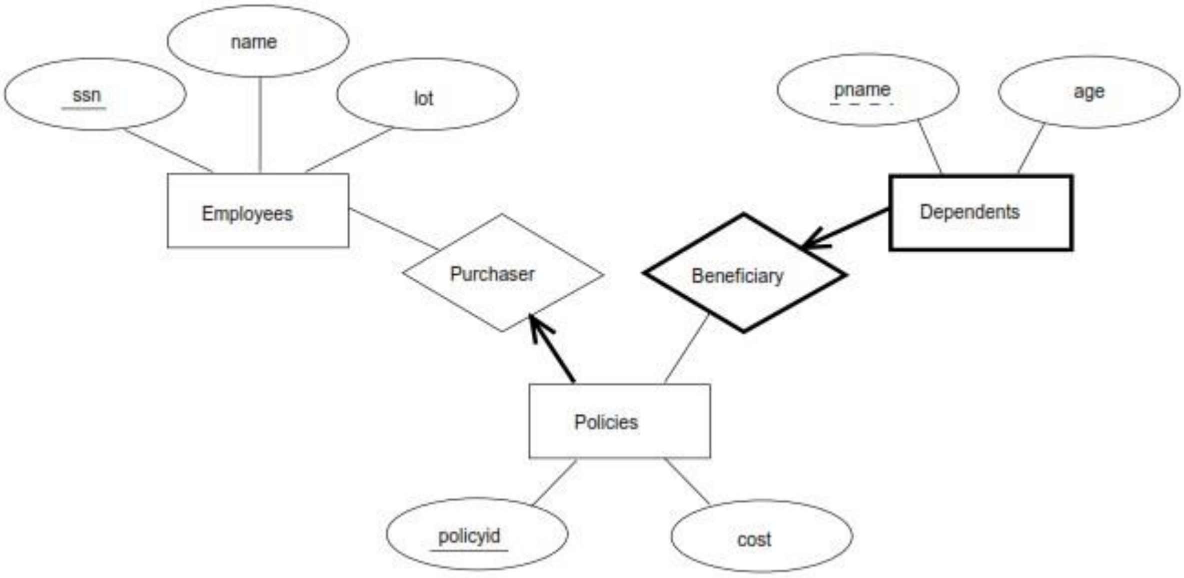
1. We have to record the descriptive attributes (in our example, since) of the Sponsorsrelationship.

2. Not every sponsorship has a monitor, and thus some <pid, did> pairs in the Sponsorsrelation may not appear in the Monitors relation.

However, if Sponsors has no descriptive attributes and has total participation in Monitors,every possible instance of the Sponsors relation can be obtained by looking at the <pid, did> columns of the Monitors relation. Thus, we need not store the Sponsorsrelation in this case.

ER to Relational: Additional Examples

Consider the ER diagram shown in below Figure.



We can translate this ER diagraminto the relational model as follows, taking advantage of the key constraints to combinePurchaser information with Policies and Beneficiary information with Dependents:

CREATE TABLE Policies (policyid INTEGER,

cost REAL,

ssn CHAR(11) NOT NULL,

PRIMARY KEY (policyid),

FOREIGN KEY (ssn) REFERENCES Employees

ON DELETE CASCADE);

```
CREATE TABLE Dependents ( pname CHAR(20),
                          age INTEGER,
                          policyid INTEGER,
                            PRIMARY KEY (pname, policyid),
                            FOREIGN KEY (policyid) REFERENCES Policies
                            ON DELETE CASCADE);
```

Notice how the deletion of an employee leads to the deletion of all policies owned bythe employee and all dependents who are beneficiaries of those policies. Further, eachdependent is required to have a covering policy, because policyid is part of the primarykey of Dependents, there is an implicit NOT NULL constraint. This model accuratelyreflects the participation constraints in the ER diagram and the intended actions whenan employee entity is deleted.In general, there could be a chain of identifying relationships for weak entity sets.

Ex: we assumed that policyid uniquely identifies a policy. Suppose that policyidonly distinguishes the policies owned by a given employee; that is, policyid is only apartial key and Policies should be modeled as a weak entity set. This new assumptionabout policyid doesnot cause much to change in the preceding discussion. In fact,the only changes are that the primary key of Policies becomes <policyid, ss>,andasa consequence, the definition of Dependents changes-a field called ssn is added andbecomes part of both the primary key of Dependents and the foreign key referencingPolicies:

```
CREATE TABLE Dependents ( pname CHAR(20),
                          ssn CHAR(11),
                          age INTEGER,
                          policyid INTEGER NOT NULL,
                          PRIMARY KEY (pname, policyid, ssn),
                 FOREIGN KEY (policyid, ssn) REFERENCES Policies
                 ON DELETE CASCADE);
```

6. Introduction to Views

Any relation that is not a part of the logical model, but is made visible to a user a virtual relation

(or)

A View is a subset of the database sorted and displayed in a particular way.

Syntax:create view v as <query expression>

Where query expression is any legal expression. This view name is represented by 'v'

View provides several benefits.

1. Views can hide complexity

If you have a query that requires joining several tables, or has complex logic or calculations, you can code all that logic into a view, then select from the view just like you would a table.

2. Views can be used as a security mechanism

A view can select certain columns and/or rows from a table, and permissions set on the view instead of the underlying tables. This allows surfacing only the data that a user needs to see.

3. Views can simplify supporting legacy code

If you need to refactor a table that would break a lot of code, you can replace the table with a view of the same name. The view provides the exact same schema as the original table, while the actual schema has changed. This keeps the legacy code that references the table from breaking, allowing you to change the legacy code at your leisure.

Ex: Consider the Students and Enrolled relations.Suppose that we are often interested in finding the names and student identifiers ofstudents who got a grade of B in some course, together with the cid for the course.We can define a view for this purpose. Using SQL notation:

CREATE VIEW B-Students (name, sid, course)

AS SELECT S.sname, S.sid, E.cid

FROM Students S, Enrolled E

WHERE S.sid = E.sid AND E.grade = `B';

The view B-Students has three fields called name, sid,andcourse with the samedomains as the fields sname and sid in Students and cid in Enrolled.

This view can be used just like a base table, or explicitly stored table, in defining newqueries or views. Given the instances of enrolled and students tables specified above, BStudentscontains the tuples shown in below Figure. Conceptually, whenever B-Studentsis used in a query, the view definition is first evaluated to obtain the correspondinginstance of B-Students, and then the rest of the query is evaluated treating B-Studentslike any other relation referred to in the query.

name	*sid*	*course*
Jones	53666	History105
Guldu	53832	Reggae203

Views, Data Independence, Security

Consider the levels of abstraction,the physicalschema for a relational database describes how the relations in the conceptual schema are stored, in terms of the file organizations and indexes used. The conceptual schema isthe collection of schemas of the relations stored in the database. While some relationsin the conceptual schema can also be exposed to applications, i.e., be part of theexternal schema of the database, additional relations in the external schema can bedefined using the view mechanism. The view mechanism thus provides the supportfor logical data independence in the relational model. That is, it can be used to definerelations in the external schema that mask changes in the conceptual schema of thedatabase from applications.

Ex: if the schema of a stored relation is changed,we can define a view with the old schema, and applications that expect to see the oldschema can now use this view.

Views are also valuable in the context of security: We can define views that give agroup of users' access to just the information they are allowed to see.

Ex: wecan define a view that allows students to see other students' name and age but nottheir gpa, and allow all students to access this view, but not the underlying Students table.

Updates on Views

A view can be used just like any otherrelation in defining a query. However, it is natural to want to specify updates on viewsas well.

The SQL-92 standard allows updates to be specified only on views that are definedon a single base table using just selection and projection, with no use of aggregateoperations. Such views are called updatable views. This definition is oversimplified,but it captures the spirit of the restrictions. An update on such a restricted view canalways be implemented by updating the underlying base table in an unambiguous way.Consider the following view:

CREATE VIEW GoodStudents (sid, gpa)

AS SELECT S.sid, S.gpa

FROM Students S

WHERE S.gpa> 3.0;

We can implement a command to modify the gpa of a GoodStudents row by modifyingthe corresponding row in Students. We can delete a GoodStudents row by deletingthe corresponding row from Students.

We can insert a GoodStudents row by inserting a row into Students, using null valuesin columns of Students that do not appear in GoodStudents (e.g., sname, login). Notethatprimary key columns are not allowed to contain null values. Therefore, if weattempt to insert rows through a view that does not contain the primary key of theunderlying table, the insertions will be rejected.

Ex: if GoodStudents containedsname but not sid, we could not insert rows into Students through insertionsto GoodStudents.

An important observation is that an INSERT or UPDATE may change the underlyingbase table so that the resulting (i.e., inserted or modified) row is not in the view.

Ex: if we try to insert a row<51234, 2.8> into the view, this row can be (paddedwith null values in the other fields of Students and then) added to the underlyingStudents table, but it will not appear in the GoodStudents view because it does notsatisfy the view condition gpa> 3:0. The SQL-92 default action is to allow thisinsertion, but we can disallow it by adding the clause WITH CHECK OPTION to thedefinition of the view.

Need to Restrict View Updates

While the SQL-92 rules on updatable views are more stringent than necessary, thereare some fundamental problems with updates specified on views, and there is goodreason to limit the class of views that can be updated. Consider the Students relationanda new relation called Clubs:

Clubs(cname: string, jyear: date, mname: string)

A tuple in Clubs denotes that the student called mname has been a member of theclub cname since the date jyear.

Suppose that we are often interested in finding thenames and logins of students with a gpa greater than 3 who belong to at least oneclub, along with the club name and the date they joined the club. We can define aview for this purpose:

CREATE VIEW ActiveStudents (name, login, club, since)

AS SELECT S.sname, S.login, C.cname, C.jyear

FROM Students S, Clubs C

WHERE S.sname = C.mname AND S.gpa> 3;

Consider the instances of Students and Clubs shown in following Figures

cname	*jyear*	*mname*
Sailing	1996	Dave
Hiking	1997	Smith
Rowing	1998	Smith

An Instance *C* of Clubs

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8

An Instance *S3* of Students

Whenevaluated using the instances C and S3, ActiveStudents contains the rows shown Figure

name	*login*	*club*	*since*
Dave	dave@cs	Sailing	1996
Smith	smith@ee	Hiking	1997
Smith	smith@ee	Rowing	1998
Smith	smith@math	Hiking	1997
Smith	smith@math	Rowing	1998

Now suppose that we want to delete the row <Smith, smith@ee, Hiking, 1997> from ActiveStudents. How are we to do this? ActiveStudents rows are not stored explicitly butare computed as needed from the Students and Clubs tables using the view definition.So we must change either Students or Clubs (or both) in such a way that evaluating theview definition on the modified instance does not produce the row <Smith, smith@ee,Hiking, 1997.>

This task can be accomplished in one of two ways: by either deletingthe row <53688, Smith, smith@ee, 18, 3.2> from Students or deleting the row <Hiking,1997, Smith> from Clubs. But neither solution is satisfactory. Removing the Studentsrow has the effect of also deleting the row<Smith, smith@ee, Rowing, 1998> from theview ActiveStudents. Removing the Clubs row has the effect of also deleting the row<Smith, smith@math, Hiking, 1997> from the view ActiveStudents. Neither of theseside effects is desirable. In fact, the only reasonable solution is to disallow such updateson views.

There are views involving more than one base table that can, in principle, be safelyupdated. The B-Students view that we introduced at the beginning of this sectionis an example of such a view. Consider the instance of B-Students shown in above Figure. To insert a tuple, say <Dave, 50000, Reggae203> B-Students, we can simply insert a tuple <Reggae203, B, 50000> into Enrolled since there is already a tuple for sid 50000in Students. To insert <John, 55000, Reggae203> on the other hand, we have to insert<Reggae203, B, 55000> into Enrolled and also insert <55000, John, null, null, null>into Students. Observe how null values are used in fields of the inserted tuple whosevalue is not available. Fortunately, the view schema contains the primary key fields of both underlying base tables;

otherwise, we would not be able to support insertionsinto this view. To delete a tuple from the view B-Students, we can simply delete thecorresponding tuple from Enrolled.

Although this example illustrates that the SQL-92 rules on updatable views are unnecessarilyrestrictive, it also brings out the complexity of handling view updates inthe general case. For practical reasons, the SQL-92 standard has chosen to allow onlyupdates on a very restricted class of views.

7. Destroying/Altering Tables and Views

If we decide that we no longer need a base table and want to destroy it (i.e., deleteall the rows and remove the table definition information), we can use the DROP TABLEcommand.

Ex: DROP TABLE Students RESTRICT destroys the Students tableunless some view or integrity constraint refers to Students; if so, the command fails.

If the keyword RESTRICT is replaced by CASCADE, Students is dropped and any referencingviews or integrity constraints are (recursively) dropped as well; one of thesetwo keywords must always be specified.

A view can be dropped using the DROP VIEWcommand, which is just like DROP TABLE.

ALTER TABLE modifies the structure of an existing table. To add a column calledmaiden-name to Students, for example, we would use the following command:

ALTER TABLE StudentsADD COLUMN maiden-name CHAR(10);

The definition of Students is modified to add this column, and all existing rows arepadded with null values in this column. ALTER TABLE can also be used to deletecolumns and toadd or drop integrity constraints on a table;