Lect#5: SQL

Ref1 for STUDENT RECORD DB:

"Database Design and Implementation"

Edward Sciore, Boston College

ISBN: 978-0-471-75716-0

Ref2 for COMPANY DB:

"Fund. of Database Systems",

Elmasri, Navathe, 5th ed., Addison Wesley

SQL (*Structured Query Language*)

- History:
  - SQL1, @1989
  - SQL2=SQL92 *(create schema, referential integrity options and additional data types (s.a.* DATE, TIME, and TIMESTAMP*) are added..)*
  - SQL3=SQL99 *(object-relational database concepts, call level interfaces, and integrity management)*
- *SQL is based on*
  - *Relational algebra:*
    - *introduced by E. F. Codd in 1972,*
    - *Provide the basic concepts behind computing SQL syntax.*
    - *It is a procedural way to construct data-driven queries*
    - *it addresses the* ***how*** *logic of a structured query.*
- Additionally SQL provides
  - insertion, modification and deletion.
  - Arithmetic operators
  - Display of data
  - Assignment
  - Aggregate functions

SQL: Data Definition, Constraints, and Schema Changes

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database
- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- Specifying constraints
  - NOT NULL may be specified on an attribute
  - Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases
  - referential integrity constraints (foreign keys).

```
CREATE TABLE DEPT (
   DNAME    VARCHAR(10)   NOT NULL,
   DNUMBER  INTEGER       NOT NULL,
   MGRSSN   CHAR(9)  DEFAULT '000',
   MGRSTARTDATE        CHAR(9),
   PRIMARY KEY (DNUMBER),
   UNIQUE (DNAME),
   FOREIGN KEY (MGRSSN) REFERENCES EMP
      ON DELETE SET DEFAULT
      ON UPDATE  CASCADE);
```

```
CREATE TABLE EMP(
   ENAME     VARCHAR(30)NOT NULL,
   ESSN      CHAR(9),
   BDATE     DATE,
   DNO       INTEGER   DEFAULT 1,
   SUPERSSN  CHAR(9),
   PRIMARY KEY (ESSN),
   FOREIGN KEY (DNO) REFERENCES   DEPT ON
   DELETE SET DEFAULT ON UPDATE  CASCADE,
   FOREIGN KEY (SUPERSSN) REFERENCES EMP
   ON DELETE SET NULL ON UPDATE CASCADE);
```

cont.

- **Integrity Constraints in Create Table**
  - **not null**
  - **primary key**(*A*1, ..., *An*)
  - **foreign key** (*A*m, ..., *An*) **references** *r*
- **primary key** declaration on an attribute automatically ensures **not null**
- Some foreign keys may cause errors
  - Specified either via:
    - Circular references, or
    - they refer to a table that has not yet been created. *(Discussed previously in relationl model constraints)*

```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber           INT             NOT NULL,
          Dlocation         VARCHAR(15)     NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname             VARCHAR(15)     NOT NULL,
          Pnumber           INT             NOT NULL,
          Plocation         VARCHAR(15),
          Dnum              INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn              CHAR(9)         NOT NULL,
          Pno               INT             NOT NULL,
          Hours             DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn              CHAR(9)         NOT NULL,
          Dependent_name    VARCHAR(15)     NOT NULL,
          Sex               CHAR,
          Bdate             DATE,
          Relationship      VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

cont. (more detailed create command)

```
CREATE TABLE EMPLOYEE
    (  . . . ,
       Dno        INT             NOT NULL        DEFAULT 1,
    CONSTRAINT EMPPK
       PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
       FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
       FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    (  . . . ,
       Mgr_ssn    CHAR(9)         NOT NULL        DEFAULT '888665555',
       . . . ,
    CONSTRAINT DEPTPK
       PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
       UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
       FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET DEFAULT   ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    (  . . . ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE CASCADE           ON UPDATE CASCADE);
```

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

DROP TABLE / ALTER TABLE

- DROP TABLE
  - Used to remove a relation (base table) and its definition
  - The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
  - Example: **DROP TABLE DEPENDENT;**

- ALTER TABLE
  - Used to add an attribute to one of the base relations
    - The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute
  - Example:
    ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);
  - The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple.
    - This can be done using the UPDATE command.

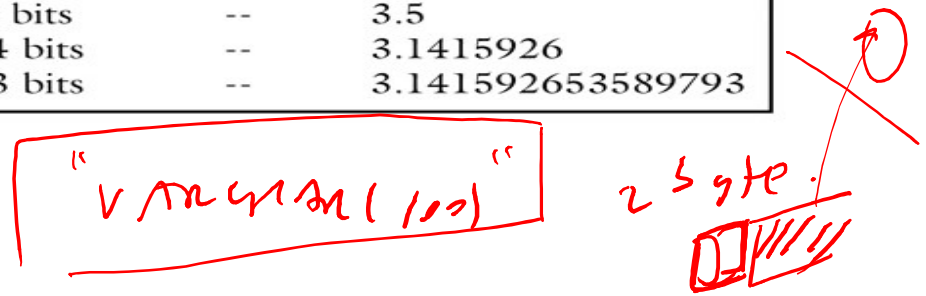
Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
  - This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples. Example: {A, B, C, A} is a bag. {A, B, C} is also a bag that also is a set. Bags also resemble lists, but the order is irrelevant in a bag.
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

SQL Types:numeric,string,date

Type Name	Flavor	Precision	Scale	Sample Constant
NUMERIC(5,3)	exact	5 digits	3	31.416
INTEGER or INT	exact	9 digits	0	314159265
SMALLINT	exact	4 digits	0	3142
FLOAT(3)	approximate	3 bits	--	3.5
FLOAT	approximate	24 bits	--	3.1415926
DOUBLE PRECISION	approximate	53 bits	--	3.141592653589793

Figure 4-14
Numeric types in SQL



VARCHAR(n), CHAR(n)

DATE '2008-07-03' = 3 July,2008

INTERVAL '5' DAY

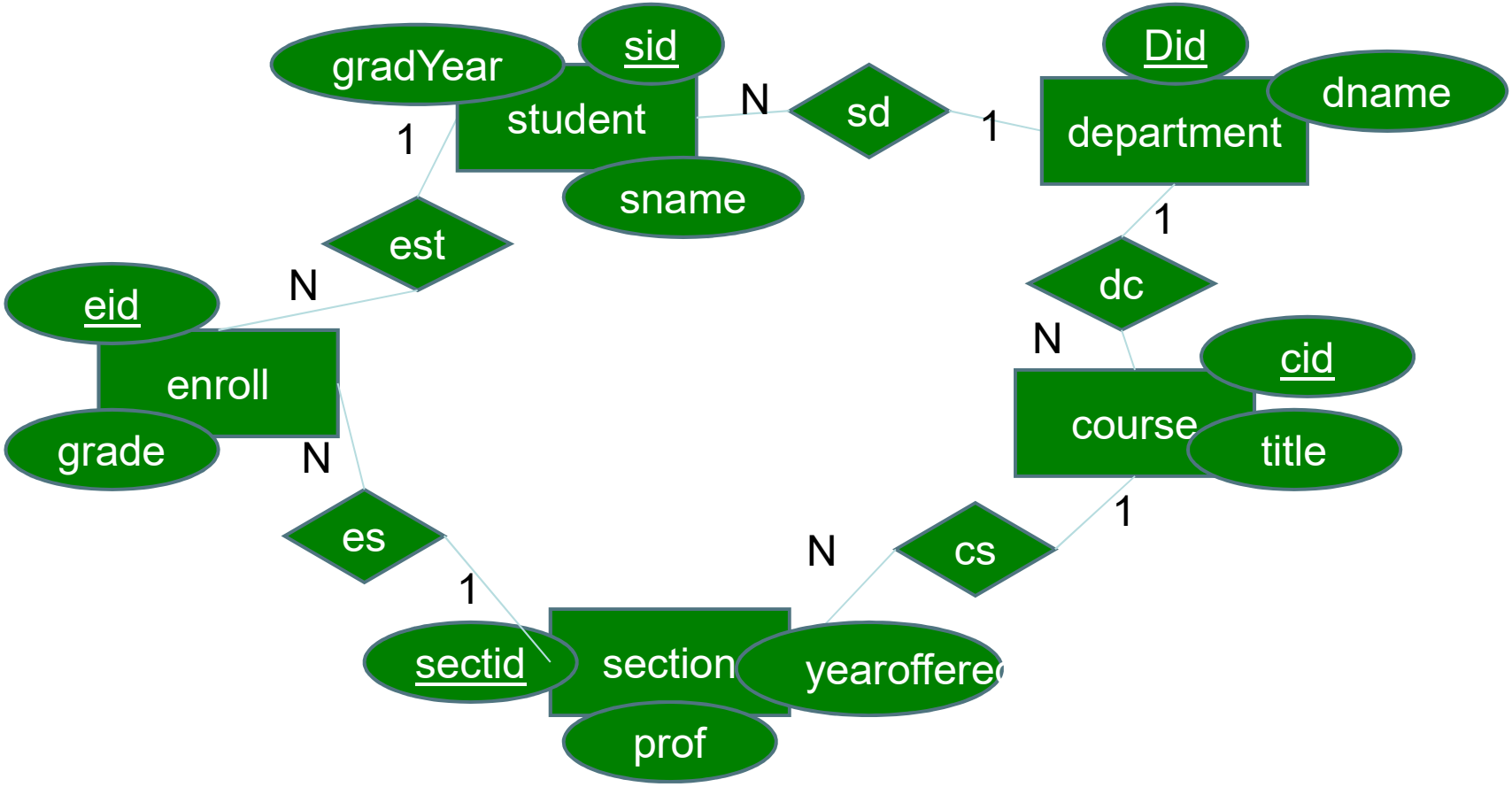
Function Name	Meaning	Example Usage	Result
current_date	Return the current date.	`current_date`	`Today's date`
extract	Extract the year, month, or day from a date.	`extract(month, date '2008-07-04')`	`7`
+	Add an interval to a date.	`date '2008-07-04' + interval '7' month`	`date '2009-02-04'`
-	Subtract an interval from a date or subtract two dates.	`date '2008-07-04' - date '2008-06-30'`	`interval '5' day`

igure 4-16
ome common SQL date/interval functions

Function Name	Meaning	Example Usage	Result
lower (also, *upper*)	Turn the characters of the string into lower case (or upper case).	`lower('Einstein')`	`'einstein'`
trim	Remove leading and trailing spaces.	`trim(' Einstein ')`	`'Einstein'`
char_length	Return the number of characters in the string.	`char_length ('Einstein')`	`8`
substring	Extract a specified substring.	`substring('Einstein' from 2 for 3)`	`'ins'`
current_user	Return the name of the current user.	`current_user`	`'einstein'` (assuming that he is currently logged in)
||	Catenate two strings.	`'A. ' || 'Einstein'`	`'A. Einstein'`
like	Match a string against a pattern.	`'Einstein' like '_i%i_%'`	`true`

Figure 4-15

A student record database:




```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

Student record db state

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

COURSE	CId	Title	DeptId
	12	db systems	10
	22	compilers	10
	32	calculus	20
	42	algebra	20
	52	acting	30
	62	elocution	30

SECTION	SectId	CourseId	Prof	YearOffered
	13	12	turing	2004
	23	12	turing	2005
	33	32	newton	2000
	43	32	einstein	2001
	53	62	brando	2001

ENROLL	EId	StudentId	SectionId	Grade
	14	1	13	A
	24	1	43	C
	34	2	43	B+
	44	4	33	B
	54	4	53	A
	64	6	53	A

Figure 1-1
Some records for a university database

select

- Calculate a new Id and the graduation decade for each student
  - Q68: select 'Student #' || s.SId AS NewSId, s.SName,
    cast(s.GradYear/10 as int) * 10 AS GradDecade
    from STUDENT s

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

Q68	NewSId	SName	GradDecade
	Student #1	joe	2000
	Student #2	amy	2000
	Student #3	max	2000
	Student #4	sue	2000
	Student #5	bob	2000
	Student #6	kim	2000
	Student #7	art	2000
	Student #8	pat	2000
	Student #9	lee	2000

- **Calculate the number of years since graduation for each student**
  - Q69: select s.*, extract(YEAR,current_date)-s.GradYear AS AlumYears
    from STUDENT s;
- **Determine if a student has graduated**
  - Q71: select q.*, if(q.AlumYrs>0,'alum', 'in school') AS GradStats
    from Q69 q;

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

from

- **All combinations of STUDENT and DEPT records..**
  - Q73: select s.*,d.*
    from STUDENT s,DEPT d

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

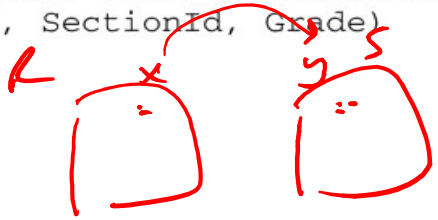
- **All pairs of STUDENT names**
  - Q74: select s1.SNAME as name1, s2.SName as name2
    from STUDENT s1, STUDENT s2
- **The join of STUDENT and DEPT**
  - Q75: select s.SName, d.DName
    from STUDENT s **join** DEPT d ON s.MajorId=
  - Outer join lists all records even it has no match.
    select s.SName, d.DName
    from STUDENT s **full join** DEPT d ON s.MajorId=d.DId

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

- **The names of students and their professors**
  - Q76: select s.SName, k.Prof
    from (STUDENT s join ENROLL e ON s.SId=e.StudentId) join
    SECTION k on e.SectionId = k.SectId

where

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```



- **The names of students graduating in 2005 or 2006.**
  - Q77: Select s.SName
    From STUDENT s
    Where (s.GradYear=2005) or (s.GradYear=2006)
- **The names of students and their professors *(compare with Q76)***
  - Q78: select s.SName, k.Prof
    from STUDENT s, ENROLL e,SECTION k
    where s.SId=e.StudentId and e.SectionId = k.SectId

- **Find the grades Joe received during his graduation year. *(remember Q29)***
  - Q79: select e.Grade
    from STUDENT s, ENROLL e,SECTION k
    where s.SId=e.StudentId and e.SectionId = k.SectId and
    k.YearOffered=s.GradYear and s.SName='Joe';
- **The students whose names begin with 'j' and graduate this year**
  - Q80: select s.*
    from STUDENT s
    where s.SName like 'j%' and s.GradYear=extract(YEAR,current_date)
- **Grades given by the current professor-user**
  - Q81: select s.SName, e.Grade
    from STUDENT s, ENROLL e,SECTION k
    where s.SId=e.StudentId and e.SectionId = k.SectId and
    k.Prof=current_user;

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

groupby

1. Grouping fields are in group by clause. Aggragate functions are in select clause
2. The order of execution:
   FROM ➔ WHERE ➔ GROUP BY ➔ SELECT
3. The only non-computed fields allowed in the select are grouping fields.
4. Thus; empty group by clause AND select clause with aggragate function can only show the result of aggragate function.

- **The minumum and maximum graduation year per major *(remember Q12)***
  - Q82: select s.MajorId, min (GradYear), max (GradYear)
    from STUDENT s
    group by s.MajorId
- **Find the section that gave out the most 'A' grades. *(remember Q18-20)***

Q83: select e.SectionId, count (EId) AS numAs
from ENROLL e
where e.Grade='A'
group by e.SectionId

The maximum number of A's given in any section:
Q84: select max (q.numAs) AS maxAs
from Q83 q

Find the section that gave out the most 'A' grades.
Q85: select Q83.SectionId
from Q83, Q84
where Q83.numAs=Q84.maxAs

illegal

Find the section that gave out the most 'A' grades
Q84a: select q.SectionId, max (q.numAs) as maxAs
from Q83 q

Find the section that gave out the most 'A' grades.
Q84b: select q.SectionId
from Q83 q
where q.numAs=max (q.numAs)

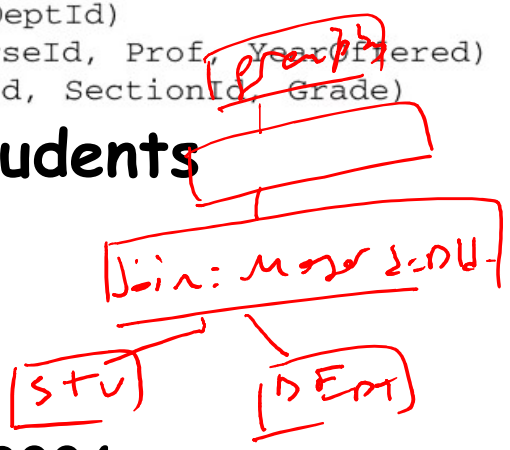
group by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Show the name of each student and the number of courses they took.
  - Q86: select e.StudentId, count (EId) AS HowMany
    from ENROLL e
    group by e.StudentId
    Q87: select s.SName,q.HowMany
    from Q86 q, STUDENT s
    where q.StudentId=s.StudentId
  - Q87 is technically correct. But does not show students that have not yet taken any course.
    Thus, we need outerjoin:
    Q87a: select s.SName,q.HowMany
    from STUDENT s **full join** Q86 q ON q.StudentId=s.~~StudentId~~ SId
  - Q87a is technically correct. But does not show 0 number of course for students that have not yet taken any course. It shows NULL. Thus, we do outerjoin before counting.
    Q88: select e.SName, count (EId) AS HowMany
    from STUDENT s **full join** ENROLL e ON s.SId=e.StudentId
    group by s.SName
  - Q88 is technically correct. But it groups records by student name in order to show it in the output. In case of any student having the same name,the result will be erroneous.
    Q89: select e.SName,count (e.EId) AS HowMany
    from STUDENT s **full join** ENROLL e ON s.SId=e.StudentId
    group by s.SId, s.SName *// has no difference with group by s.SId*

group by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```



- **List of all major department names of students graduated in 2004, with no duplicates.**
  - Q90: select d.DName
    from STUDENT s, DEPT d
    where s.MajorId = d.DId and s.GradYear=2004
    group by d.DName *// used to remove duplicates!*
  - *//alternate version of Q90*
    Q91: select **distinct** d.DName
    from STUDENT s, DEPT d
    where s.MajorId = d.DId and s.GradYear=2004
- **Show the number of different majors that students have taken.**
  - Q92: select count (distinct s.MajorId)
    from STUDENT s

having

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- List professors who taught more than 4 sections in 2008.
  - Q93: select k.Prof, count(k.SectId) as howMany
    from SECTION k
    where k.YearOffered=2008 and howMany>4
    group by k.Prof

    illegal
  - Q94: select k.Prof, count(k.SectId) as howMany
    from SECTION k
    where k.YearOffered=2008
    group by k.Prof

    Q95: select q.Prof, q.howMany
    from Q94 q
    where q.howMany>4
  - Q96: select k.Prof, count(k.SectId) as howMany
    from SECTION k
    where k.YearOffered=2008
    group by k.Prof
    having count(k.SectId)>4
- The order of execution:
  FROM ➔ WHERE ➔ GROUP BY ➔ HAVING ➔ SELECT

Nested queries

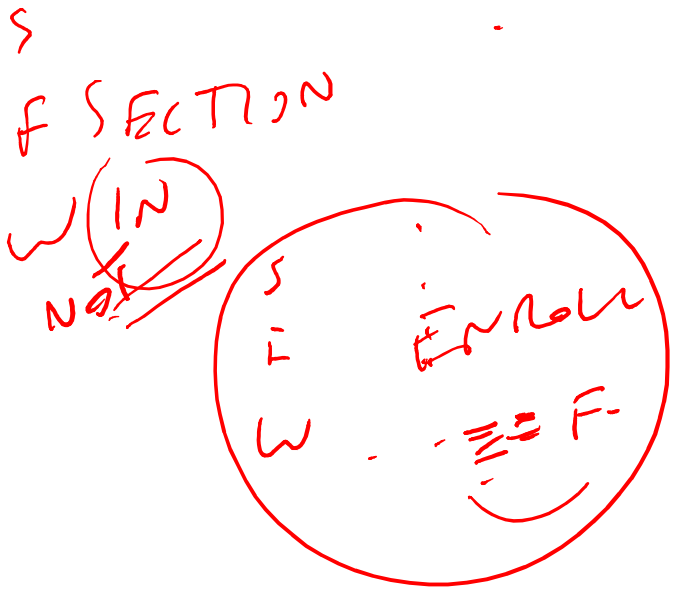
```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- Nested queries is used to express semijoin and antijoin.
- (T1 $\ltimes_{A=B}$ T2) ≡ select * from T1 where A **IN** (select B from T2)
- (T1 $\triangleright_{A=B}$ T2) ≡ select * from T1 where A **NOT IN** (select B from T2)
- Determine the departments having at least one major *(remember Q35)*
  - Q97: select d.*
    from DEPT d
    where d.DId IN (select s.MajorId
    from STUDENT s)
- Determine those students who took a course with Prof. Einstein. *(remember Q38-40)*
  - Q98: select s.*
    from STUDENT s
    where s.SId IN
    (select e.StudentId
    from ENROLL e
    where e.SectionId IN
    (select k.SectId
    from SECTION k
    where k.Prof='einstein'))

Nested queries:

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- **List the sections where "nobody" received an 'F'. *(remember Q42-43)***
  - Q99: select
    from SECTION k
    where k.SectId NOT IN
    (select e.SectionId
    from ENROLL e
    where e.Grade ='F')



union:

- **Combined names of students and professors *(remember Q54)***
  - Q100: select s.SName as Person from STUDENT s
    union
    select k.Prof as Person from SECTION k

order by

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- The order of execution:

  FROM ➔ WHERE ➔ GROUP BY ➔ HAVING ➔ SELECT ➔ UNION ➔ORDER BY

- The number of math majors per graduation year sorted by count descending and then year.
  - Q101: select s.GradYear, count(SId) as howMany
    from STUDENT s
    group by s.GradYear
    order by howMany DESC, s.GradYear
- The combined names of students and professors in sorted order *(Sorted version of Q100)*
  - Q100: select s.SName as Person from STUDENT s
    union
    select k.Prof as Person from SECTION k
    order by Person

SQL update

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

1. insert 1 new record by giving its values
2. insert multiple new record by specifying a query
3. Delete records
4. Modify the contents of records

- insert into STUDENT (SId,SName,MajorId,GradYear)
  values (10, 'ron', 30, 2009)

- create table ALUMNI(SId int,SName varchar(10),Major varchar(8),GradYear int)
  insert into ALUMNI (SId, SName, Major, GradYear)
  select s.SId , s.SName, d.DName , s.GradYear
  from STUDENT s, DEPT d
  where s.MajorId = d.DId and s.GradYear<extract (YEAR, current_date)

- insert into STUDENT (SId , SName, Major , GradYear)
  select 22 AS SId , 'jon' AS SName, s.MajorId , s.GradYear
  from STUDENT s
  where s.SId=1

1. delete from SECTION where SectID NOT IN (select e.SectionId from ENROLL e)
2. delete from SECTION

- update STUDENT set MajorId=10, GradYear=GradYear+1 where MajorId=20

COMPANY Database Schema and state--Figure 5.5 (from Elmasri/Navathe)

All following examples use the COMPANY database

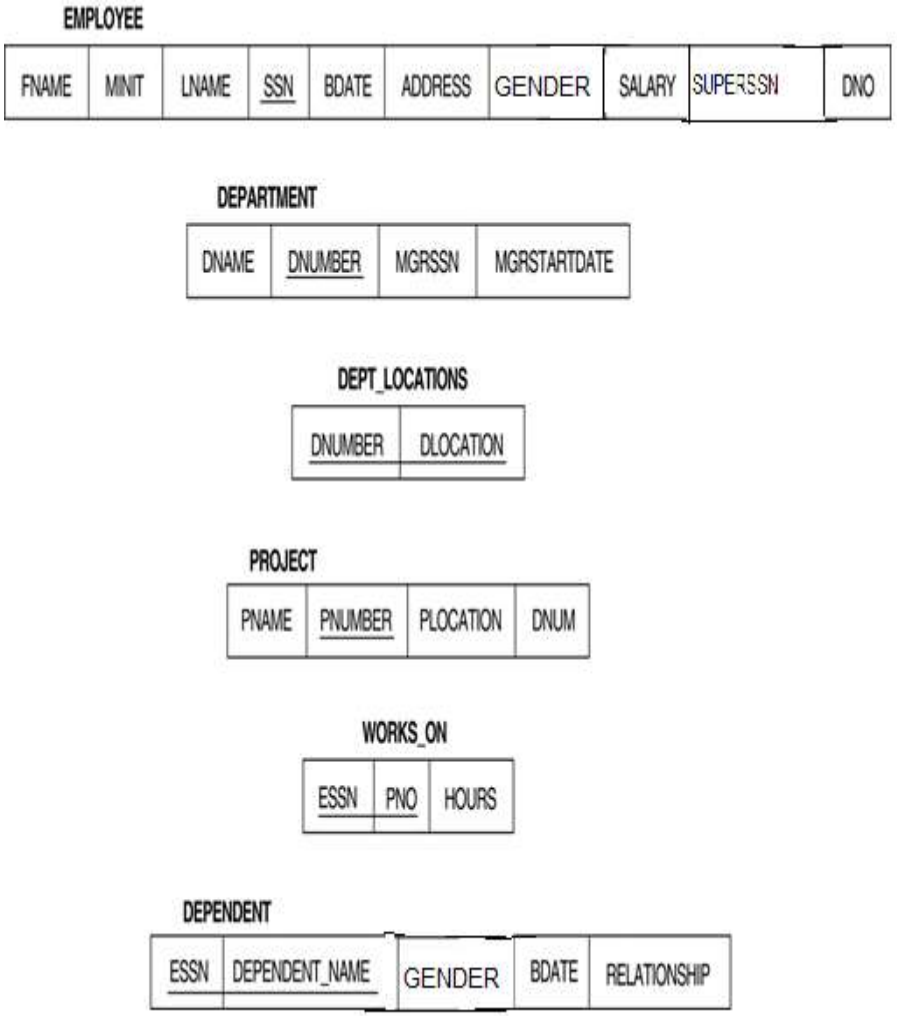


Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Gen	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

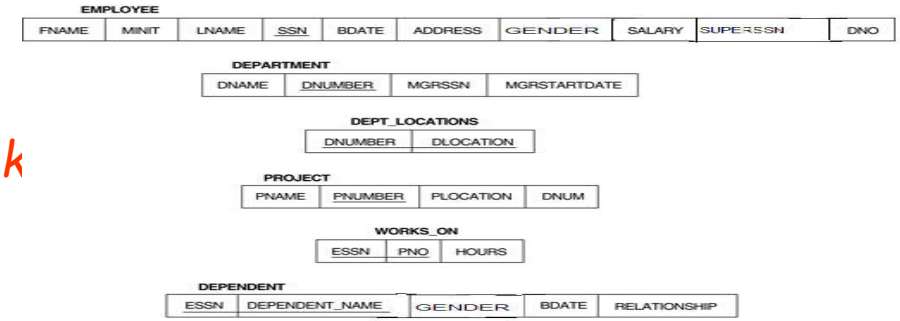
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Gen	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Sample Queries on COMPANY

(Query numbers are according to textbook

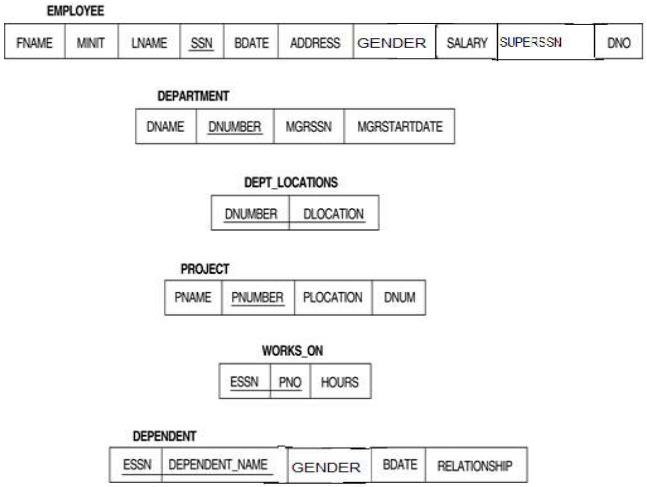


- Query 14: Retrieve the names of all employees who do not have supervisors.
  - Q14:
```
SELECT  FNAME, LNAME
FROM            EMPLOYEE
WHERE  SUPERSSN IS NULL
```
- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX' in it.

Q25:
```
SELECT  FNAME, LNAME
FROM    EMPLOYEE
WHERE ADDRESS LIKE  '%Houston,TX%'
```
- Query 26: Retrieve all employees who were born during the 1950s.
  - *Here, '5' must be the 8th character of the string (according to our format for date),*

Q26:
```
SELECT  FNAME, LNAME
FROM            EMPLOYEE
WHERE  BDATE LIKE       '_______5_'
```

Sample Queries on COMPANY



- **Query 1: Retrieve the name and address of all employees who work for the 'Research' department.**

```
Q1:  SELECT   FNAME, LNAME, ADDRESS
     FROM     EMPLOYEE
     WHERE    DNO IN (SELECT DNUMBER
                      FROM   DEPARTMENT
                      WHERE  DNAME='Research' )
```

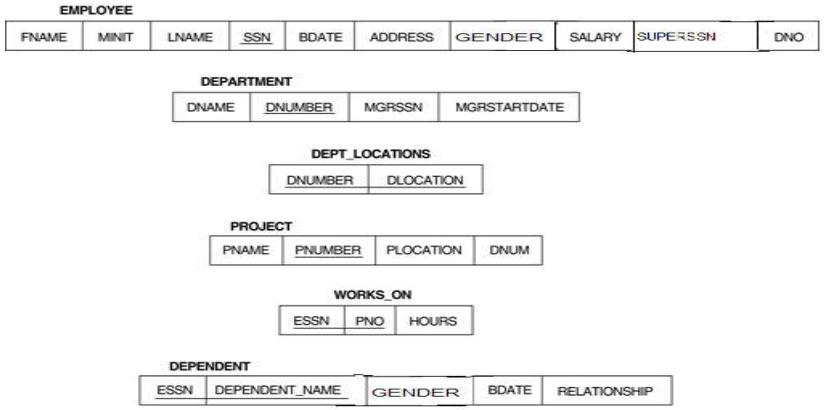
- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*
  - The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) of the outer query
- **Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.**

```
Q12: SELECT   E.FNAME, E.LNAME
     FROM     EMPLOYEE AS E
     WHERE    E.SSN IN
                    (SELECT ESSN
                     FROM   DEPENDENT
                     WHERE  ESSN=E.SSN AND
                            E.FNAME=DEPENDENT_NAME)
```

A query written with nested SELECT... FROM... WHERE... blocks and using **the = or IN** comparison operators can ***always*** be expressed as a single block query.

```
Q12A:    SELECT E.FNAME, E.LNAME
         FROM   EMPLOYEE E, DEPENDENT D
         WHERE  E.SSN=D.ESSN AND
                E.FNAME=D.DEPENDENT_NAME
```

Sample Queries on COMPANY



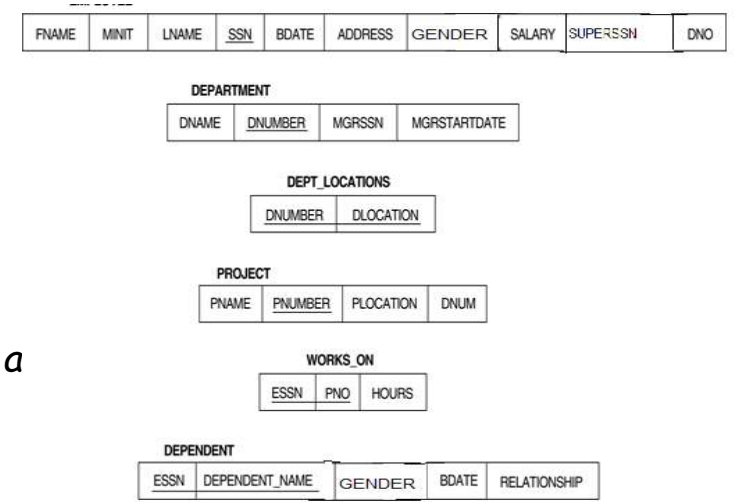
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B:   SELECT  FNAME, LNAME
        FROM    EMPLOYEE
        WHERE EXISTS  (SELECT       *
                       FROM         DEPENDENT
                       WHERE        SSN=ESSN AND
                                FNAME=DEPENDENT_NAME)
```

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:  SELECT FNAME, LNAME
     FROM   EMPLOYEE
     WHERE NOT EXISTS  (SELECT *
                        FROM         DEPENDENT
                        WHERE        SSN=ESSN)
```

Sample Queries on COMPANY



- Query 3: Retrieve the name of each employee who works all the projects on controlled by department number 5.

  Q3 (rephrase) select each employee s.t. there does not exist a project controlled by dept.#5 that the employee that the employee does not work on.

```
SELECT FNAME, LNAME
FROM   EMPLOYEE
WHERE NOT EXIST
          (SELECT *
           FROM WORKS_ON B
           WHERE (B.PNO IN (SELECT PNUMBER
                            FROM PROJECT
                            WHERE DNUM=5))
           AND
           NOT EXIST (SELECT *
                      FROM WORKS_ON C
                      WHERE SSN=C.ESSN and
                            C.PNO=B.PNO) );
```

- **Query 7: List the names of managers who have at least one dependent.**

```
Q7:     SELECT FNAME, LNAME
        FROM       EMPLOYEE
        WHERE  EXIST(SELECT *
                     FROM    DEPENDENT
                     WHERE   SSN=ESSN)
               AND
               EXIST(SELECT *
                     FROM DEPARTMENT
                     WHERE SSN=MGRSSN) )
```

Rewrite this query using only a single nested query or no nested query.

Query 22: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT PNUMBER, PNAME, COUNT(*)
        FROM   PROJECT, WORKS_ON
        WHERE  PNUMBER=PNO
        GROUP BY       PNUMBER, PNAME
        HAVING COUNT (*) > 2
```

Updates in COMPANY DB

- Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:     UPDATE      PROJECT
        SET   PLOCATION = 'Bellaire',
              DNUM = 5
        WHERE       PNUMBER=10
```

- Give all employees in the 'Research' department a 10% raise in salary.

```
U6:     UPDATE EMPLOYEE
        SET SALARY = SALARY *1.1
        WHERE DNO  IN (SELECT       DNUMBER
                       FROM         DEPARTMENT
                       WHERE        DNAME='Research')
```

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT      <attribute list>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY   <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY   <attribute list>]
```

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- The order of execution:

  FROM ➔ WHERE ➔ GROUP BY ➔ HAVING ➔ SELECT ➔ UNION ➔ ORDER BY

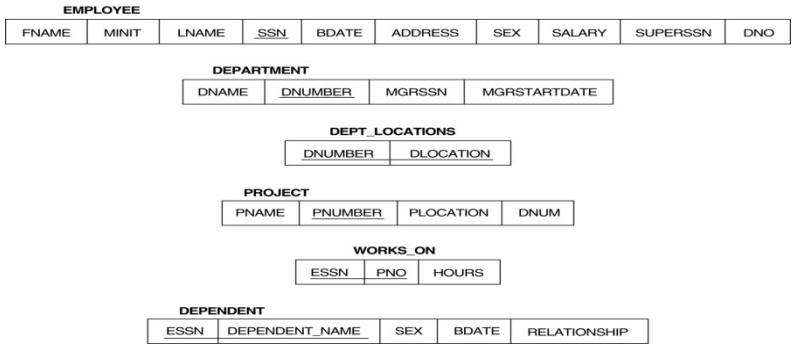
SQL views

- **Regular** View:
  - named query
  - Store only definition, real data is not stored...
  - Used to hold subqueries for convenience AND logical data independence.
- Another tyoe of View : **Materilized** View
  - Involves physically creating and keeping a temporary table
  - Disadvantage : Maintaining correspondence between the base table and the view when the base table is updated
- Ex. on Student DB:

```
create view Q65 as
select s.SName, s.GradYear
from STUDENT s
```

-
```
select q.SName
from Q65 q
where q.GradYear=2004
```

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

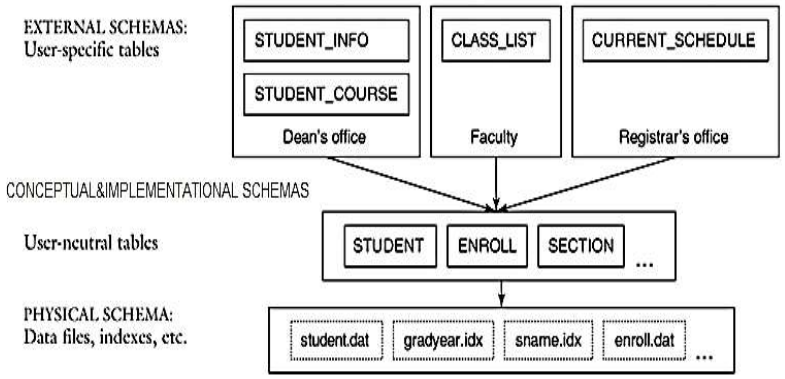


- Ex. on Student DB:

```
CREATE VIEW WORKS_ON_NEW AS
        SELECT FNAME, LNAME, PNAME, HOURS
          FROM EMPLOYEE, PROJECT, WORKS_ON
         WHERE SSN=ESSN AND PNO=PNUMBER
SELECT FNAME, LNAME FROM WORKS_ON_NEW WHERE PNAME='Seena';
DROP WORKS_ON_NEW;
```

View usage for logical data independence

- Views are used for logical data independence.
- External schema consists of views.




```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
COURSE(CId, Title, DeptId)
SECTION(SectId, CourseId, Prof, YearOffered)
ENROLL(EId, StudentId, SectionId, Grade)
```

- **STUDENT_INFO** (SId, SName,GPA, NumCoursesPassed, NumCoursesFailed)
- **STUDENT_COURSES** (SId,YearOffered, Title, Prof,Grade)
- create view **STUDENT_COURSES** as

```
select e.StudentId as SId, k.YearOffered, c.Title,
k.Prof, e.Grade
from ENROLL e, SECTION k, COURSE c
where e.SectionId=k.SectId and k.CourseId=c.CId
```

Regular View Updates

- The records in the view do not really exist, they are computed from other records in db.
- If every record *r* has a unique corresponding record *r'* in some underlying db table, then **view is updatable**.
- In general;
  - Single table views except those containing grouping are **updatable**.
  - Multi-table views are **NOT updatable.**

DEPT	DId	DName
	10	compsci
	20	math
	30	drama

STUDENT	SId	SName	GradYear	MajorId
	1	joe	2004	10
	2	amy	2004	20
	3	max	2005	10
	4	sue	2005	20
	5	bob	2003	30
	6	kim	2001	20
	7	art	2004	30
	8	pat	2001	20
	9	lee	2004	10

- create view StudentMajor as

```
select s.SName, d.DName
 from STUDENT s, DEPT d
 where s.MajorId=d.DId
```

  - Insert record {'joe', 'drama'}
    - Insert a new 'joe' record into STUDENT with having a MajorId=null. Insert a new 'drama' record into DEPT with having a DId= null.
    - Update 'joe's MajorId to 30
    - Insert a new 'joe' record into STUDENT with having a MajorId=10 *(this is valid because SName is not key)*
  - Delete record {'sue', 'math'}
    - Delete 'sue' record from STUDENT
    - Delete 'math' record from DEPT
    - Update 'MajorId' of 'sue'record in STUDENT
- Since db update via views is awkward, STORED PROCEDURES are more prevalent.