Customer(customer-name,street,city)
Branch(branch-name,account)
Account(customer-name,branch-name,account-number)
(a)Find the names of all customers who have an account in the 'Region12' branch→
$\pi_{\text{customer-name}}(\sigma_{\text{branch-name='Region12'}}(\text{Account}))$
(b)Find the names of all customers who have an account in a branch NOT located in the same city that they live in→
$\pi_{\text{customer-name}}(\sigma_{\text{A.city<>B.city^A.branch-name=B.branch-name}}(\rho_B(\text{Branch})\text{X } \rho_A(\text{Customer} \bowtie \text{Account})))$
(c)Find the branches that do not have any accounts→
$\pi_{\text{branch-name}}(\text{Branch}) — \pi_{\text{branch-name}}(\text{Account})$
(d)Find the customer names who do not have any account in the 'Region12' branch→
$\pi_{\text{customer-name}}(\text{Customer}) — \pi_{\text{customer-name}}(\sigma_{\text{branch-name='Region12'}}(\text{Account}))$
(e)Find the customer names who have accounts in all the branches located in 'Los Angeles' →
$\pi_{\text{customer-name}}(\text{Customer}) — \pi_{\text{customer-name}}(\pi_{\text{customer-name}}(\text{Customer}) \text{x } \pi_{\text{branch-name}}(\sigma_{\text{city='Los Angeles'}}(\text{Branch})) — \pi_{\text{customer-name,branch-name}}(\text{Account}))$
(f) Find the customer names who have only one account →
$\pi_{\text{customer-name}}(\text{Customer}) — \pi_{\text{A.customer-name}}(\sigma_{\text{A.branch-name<>B.branch-nameVA.account-number<>B.account-number^A.customer-name=B.customer-name}}(\rho_A(\text{Account}) \text{ x } \rho_B(\text{Account})))$

Student(sid,GPA)
Find the ids of the students with the lowest GPA
$\pi_{\text{sid}}(\text{Student}) — \pi_{\text{A.sid}}(\sigma_{\text{A.GPA>B.GPA^A.sid<>B.sid}}(\rho_A(\text{Student}) \text{ x } \rho_B(\text{Student})))$

Employee(person-name,age,street,city)
Work(person-name,company-name,salary)
Company(company-name,city)
Manage(person-name,manager-name)
(a)Find the name(s) of the employee(s) whose total salary is higher than those of all employees living in Los Angeles →SELECT person-name FROM Work GROUP BY person-name HAVING SUM(salary) >ALL (SELECT SUM(salary) FROM Work W, Employee E WHERE W.person-name=E.person-name AND city='Los Angeles' GROUP BY W.person-name);
(b)Find the names of the manager(s) whose total salary is higher than that of at least one employee that they manage → SELECT person-name FROM Work W1 GROUP BY person-name HAVING SUM(salary) >SOME (SELECT SUM(salary) FROM Work W2, Manage M WHERE W1.person-name=M.manager-name AND M.person-name=W2.person-name GROUP BY W2.person-name)
ComputerProduct(manufacturer,model,price)
Desktop(model,speed,ram,hdd)
Laptop(model,speed,ram,hdd,weight)
(a)Using two INSERT statements, insert a desktop computer manufactured by HP, with model number 1200, price $1000, speed 1.2Ghz, 256 MB RAM, and an 80GB hard drive → INSERT INTO ComputerProduct VALUES ('HP',1200,1000); INSERT INTO Desktop VALUES (1200, 1.2, 256, 80);
(b)Using two DELETE statements, delete all desktops manufactured by IBM with price below $1000 → DELETE FROM Desktop WHERE model IN (SELECT model FROM ComputerProduct WHERE manufacturer='IBM' AND price <

Data Integrity Enforcement
PRIMARY KEY(dept,cnum,sec)
UNIQUE(dept,sec,title)
FOREIGN KEY sid REFERENCES Student(sid)
ON DELETE/UPDATE CASCADE/SET NULL

Disks
Seek time avg: 10msec
Q:What is the overall capacity? → KB/disk (1,000,000,000KB=1TB)
Access Time=(seek time)+(rotational delay)+(transfer time)
Q:For 6000 RPM, average rotational delay? →
100 rotations/sec, 0.01sec/rotation=10msec. average is half, so 5msec
Q:6000RPM,10000sectors/track. How long to read one sector? →
10 msec/10000=0.001msec
Q:6000RPM,10000sectors/track. Average access time to read one sector? →
(seek time)(10ms)+(rotational delay)(5ms)+(transfer time)(0.001)=15.001ms
Q:6000RPM,10000sectors/track,1KB/sector →
What is the transfer rate? →Burst Transfer rate=(RPM/60)*(sectors/track)*(bytes/sector)

CHECK Constraints
Q1:Class snum should be <600 and class units should be <10 → CHECK(cnum<600 AND unit<10)
Q2:The units of all CS classes should be >3 → CHECK(dept<>'CS' OR unit>3)
Q3:Students whose GPA<2 cannot take CS class → CHECK((sid IN(SELECT sid FROM Student WHERE GPA>=2)) OR dept<>'CS')
Q4:Can we express referential integrity constraint using CHECK constraint → CHECK(sid IN(SELECT sid FROM Student));

TRIGGER Constraints
CREATE TRIGGER <name>
AFTER/BEFORE INSERT/UPDATE/DELETE ON R
REFERENCING NEW/OLD ROW/TABLE AS <referencing clause>
FOR EACH ROW/STATEMENT
WHEN(<condition>)
BEGIN
<event>
END

Creating View
CREATE VIEW ViewName (A1,A2,…) AS
SELECT/INSERT/UPDATE/DELETE <attr> ON <table>…[WITH CHECK OPTION];

GRANT/REVOKE
GRANT SELECT ON <tbl> TO '<user>' [WITH GRANT OPTION];
REVOKE SELECT ON <tbl> FROM '<user>' [CASCADE/RESTRICT];

EX: UPDATE Student SET GPA=4.0 WHERE sid IN (SELECT sid FROM Enroll WHERE dept='CS';→Privileges needed: SELECT, UPDATE ON Student; SELECT ON Enroll. DELETE FROM Student WHERE sid NOT IN (SELECT sid FROM Enroll); →Privileges needed: SELECT, DELETE ON Student, SELECT ON Enroll

Write a trigger so that when trying to add a laptop with weight larger than 5kg, the tuple is still inserted, but the value is set to NULL.
CREATE TRIGGER T AFTER INSERT ON Laptop REFERENCING NEW ROW AS nrow FOR EACH ROW WHEN nrow.weight>5 BEGIN UPDATE Laptop SET weight=NULL WHERE model=nrow.model END

UPDATE DeptInfo SET avgsalary=10000 WHERE dept='Toy'
GRANT SELECT, UPDATE ON Employees TO Joe WITH GRANT OPTION
GRANT SELECT UPDATE ON EmployeeNames TO Joe WITH GRANT OPTION

Table Access
INSERT INTO <tbl> VALUES <val> WHERE;
SELECT <val> FROM <tbl> WHERE;
UPDATE <tbl> SET <attr=val WHERE;
DELETE FROM <tbl> WHERE

Sequential vs. Random I/O
Q: How long to read 3 sequential sectors? → 10ms*(3/10000)=0.003ms
Q: How long to read 3 random sectors? (head above 1st sctr→ 10ms/10000=0.001ms+(10ms+5ms+0.001ms)+(10ms+5ms +0.001ms)=30.003ms

Equal?
1. $\pi_A(R — S) = \pi_A(R) — \pi_A(S)$→NO;(1,2),(1,3
SELECT B FROM R GROUP BY B; SELECT DISTINCT B FROM R→YES
SELECT B FROM R WHERE NOT EXISTS (SELECT * F WHERE R.B=S.B); (SELECT B FROM R) EXCEPT (SEL FROM S)→NO, R=(1,0),(2,);S={}
SELECT B FROM R R1 WHERE R <= ALL (SELECT B F R R2 WHERE R1.A<>R2.A)); SELECT MIN(B) FROM R R=(1,0),(2,0)

2. Manager: CHECK (salary > ALL (SELECT salary FR Employee WHERE E.dept=M.dept))
Employee: CHECK (salary < ALL (SELECT salary FRO Manager WHERE M.dept = E.dept));

3. 800GB=10 surfaces x 10,000 tracks x 8GB x X sec/
X = 1000 sec/track
10ms + 5ms + 10ms/1000 = 15.01 ms

4. 40B/tuple; 1,000,000 tuples. Blocks (8GB) needed =1,000,000/200tuple/block=5,000 blocks

Overall capacity: 2 platters x 2 surfaces/platter, 200 tracks/surface x 1KB/sector=1,600,000,000 KB/disk TB/disk

Terminology:
- Relations=tables
- Attributes=columns
- Tuples=rows
- Domain=type
- Data model=graph/tree model
- Schema:structure of relations(variable type)

→EX:Student(sid:int,name:varchar(9),addr:varchar(9))
- Instance=data(value)
- Keys:set of attributes that uniquely identifies a tuple

→EX:Student(sid,nameaddr)
- Set semantics:no duplicate tuples(relational algebra)
- Multi-set semantics:duplicate

Tables:
- One primary key per table
- UNIQUE for other keys
- SQL92:no NULL in primary
- DEFAULT for default

5 Steps Database Construction:
1. Domain analysis
2. Database design(E/R model, database design theory)
3. Table creation(DDL)
4. Load

Query and update(DML)

SQL Data Types:
String
Char(n)—fixed length
Varchar(n)—variable length
Number
Integer—32 bit
Decimal (5,2)—999.99
Real, double—32 bit,64 bit
Datetime
Date—'2010-01-15'
Time—'13:50:00'
Timestamp—'2010-01-15 13:50:00'

SQL Table Creation/Deletion:
```
CREATE TABLE Course (
        dept CHAR(2) NOT NULL DEFAULT 'CS',
        cnum INT NOT NULL,
        sec INT NOT NULL,
        title VARCHAR(50),
        PRIMARY KEY(dept,cnum,sec),
        UNIQUE(dept,sec,title)
);
DROP TABLE Course;
```

Loading Data (MySQL): LOAD DATA INFILE <datafile> INTO TABLE <table>

Relational Algebra:
SELECT (σ)
PROJECT (π)
CROSS PRODUCT (x)
NATURAL JOIN (⋈)
RENAME (ρ)
UNION (∪)
SET DIFFERENCE (—)
INTERSECT (∩)
DIVISION (/)

Aggregates
SUM, AVG, COUNT, MIN, MAX
- COUNT (*) counts tuples (NULLs)

GROUPBY- duplicates are removed
HAVING - used with aggregates
ORDER BY … [ASC/DESC]
- default ASC, used for looks

NULL
- aggregates are computed ignoring NULL, except COUNT (*)
- if input to an aggregate is empty, COUNT returns 0, all others return NULL

EX:
Student(sid,name,addr,age,GPA)
Class(dept,cnum,sec,unit,title,instructor)
Enroll(sid,dept,cnum,sec)

Q1:All students→Student
Q2:Students with age<18→ $\sigma_{age<18}$(Student)
Q3:Students with GPA>3.7 and age<18→ $\sigma_{GPA>3.7\wedge age<18}$(Student)
Q4:sid and GPA of all students→$\pi_{sid,GPA}$(Student)
Q5:All departments offering a class→π_{dept}(Class)
Q6:sid and GPA of students with age<18→ $\pi_{sid,GPA}(\sigma_{age<18}$(Student))
Q7:Names of students who take CS classes
→$\pi_{name}(\sigma_{Student.sid=Enroll.sid}$(Studentx($\sigma_{dept='CS'}$(Enroll))))
Q8:Names of students who take CS classes→
π_{name}(Student⋈($\sigma_{dept='CS'}$(Enroll)))
Q9:Names of students who take classes from "John Cho"→
π_{name}(Student⋈(Enroll⋈($\sigma_{instructor='John\ Cho'}$(Class))))
Q10:Names of the student pairs who live in the same address→
$\pi_{s1.name=s2.name}(\sigma_{s1.addr=s2.addr\wedge s1.sid=s2.sid}(\rho_{s1}$(Student)x ρ_{s2}(Student)))
Q11: All people's names→π_{name}(Student) ∪ $\rho_{c(name)}$($\pi_{instructor}$(Class))
Q12:Courses(dept,cnum,sec) that no one takes→ $\pi_{dept,cnum,sec}$(Class)-$\pi_{dept,cnum,sec}$(Enroll)
Q14:Instructor names who teach both CS and EE courses→
$\pi_{instructor}(\sigma_{dept='CS'}$(Class)) ∩ $\pi_{instructor}(\sigma_{dept='EE'}$(Class))
Q15:Sids of students who take every CS class→
AllSids—π_{sid}(AllSidsxCSClasses—Enroll)
π_A(R) —$\pi_A(\pi_A$(R)xS—R) = R/S

R←enroll

A (Sid)	B (ClassID)
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_2

A
a_1

S←CS Classes

B
b_1
b_2

SQL Statements:
SELECT attributes, aggregates
FROM relations
WHERE conditions
GROUP BY attributes
HAVING aggregate condition
ORDER BY attributes

Set operators (∪,∩, −)
- NULL is treated like other values here
- check NULL: IS NULL, IS NOT NULL

Arithmetic op's and comp's
- for NULL's, UNKNOWN, only true values returned

Three-valued logic:
Truth table:
- AND: U & T = U, U & F = F, U & U = U
- OR: U | T = T, U | F = U, U | U = U

LEFT/RIGHT/FULL OUTER JOIN

MySQL:
DISTINCT (to remove duplicates)
% = any string
_ = one character

Set operators:
INTERSECT, UNION, EXCEPT
- same schema for operands
- based on set semantics
- keep duplicates with 'ALL'

Subqueries:
- can rewrite them as long as we don't have negation
- with negation, we need EXCEPT

Set membership (IN, NOT IN)
- a IN R is TRUE is a is in R

Set comparision (> ALL, < SOME,)
- <> ALL ≡ NOT IN, = SOME ≡ IN

Data Modification
```
INSERT INTO Enroll VALUES (301,'CS',201,01);
DELETE FROM Student WHERE sid NOT IN (SELECT sid FROM Enroll);
UPDATE Class SET cnum=cnum+100 WHERE dept='CS';
```

Q1:Titles and instructors of all CS classes→
```
SELECT title, instructor FROM Class WHERE dept='CS';
```
Q2:Names and GPAs of all students who take CS class(es)→
```
SELECT DISTINCT name, GPA AS Grade, FROM Student S, Enroll E WHERE dept='CS' AND S.sid=E.sid;
```
Q3:All student names and GPAs who live on Wilshire→
```
SELECT name,GPA FROM Student WHERE addr LIKE '%Wilshire%';
```
Q4:People's names both students and instructors→
```
(SELECT name FROM Student) UNION (SELECT instructor-name FROM Class);
```
Q5:Sids of students who do not take any CS class→
```
(SELECT sid FROM Student) EXCEPT (SELECT sid FROM Enroll WHERE dept='CS');
```
Q6:Sides who live with student 301→
```
SELECT sid FROM Student WHERE addr=(SELECT addr FROM Student WHERE sid=301);
```
Q7:Student names who take CS classes→
```
SELECT name FROM Student S WHERE sid IN (Select sid FROM Enroll WHERE dept='CS');
SELECT name FROM Student S, Enroll E WHERE S.sid=E.sid AND dept='CS';
```
Q8:Students names who take no CS class→
```
SELECT name FROM Student S WHERE sid NOT IN (SELECT sid FROM Enroll WHERI dept='CS');
(SELECT name FROM Student) EXCEPT (SELECT name FROM Enroll E, Student S WHERE dept='CS' AND E.sid=S.sid);
```
Q9: Studetn IDs who has higher GPA than any student of age 18 or less→
```
SELECT sid FROM Student WHERE GPA>SOME(SELECT GPA FROM Student WHERE age<=18);
```
Q10:Student IDs whose GPA is higher than at least one student of age 18 or less→
```
SELECT sid FROM Student WHERE GPA>SOME(SELECT GPA FROM Student WHERE age<=18);
```
Q11:Student names who take any class→SELECT name FROM Student S WHERE EXI
```
(SELECT * FROM Enroll E WHERE S.sid=E.sid);
```
Q12:Average GPA of all students→SELECT AVG(GPA) FROM Student;
Q13:Number of students taking CS classes→
```
SELECT COUNT(DISTINCT sid) FROM Enroll WHERE dept='CS';
```
Q14:Average GPA of students who take CS classes→
```
SELECT AVG(GPA) FROM Student S, Enroll E WHERE dept='CS' AND S.sid=E.sid;
SELECT AVG(GPA) FROM Student S WHERE sid IN (SELECT sid FROM Enroll E WHEI dept='CS');
```
Q15:Average GPA for each group→
```
SELECT age, AVG(GPA) FROM Student GROUP BY age;
```
Q16:Number of classes each student takes→
```
SELECT sid, COUNT(*) FROM Enroll GROUP BY sid;
```
Q16 Using Outer Join: SELECT sid, COUNT(*) FROM Student S LEFT OUTER JOIN Enr E ON S.sid=E.sid GROUP BY sid;
Q17:Students who take two classes or more→
```
SELECT sid FROM Enroll GROUP BY sid HAVING COUNT(*)>=2;
```