Overview of SQL

- common data sublanguage employed by most relational database management systems (Oracle, Sybase, DB-2, MySQL, SQLServer, Informix, …)

DDL	DML	DCL
CREATE	SELECT	GRANT
DROP	DELETE	REVOKE
ALTER	UPDATE	
	INSERT	

- the above is the minimum & necessary instruction set for SQL; a particular implementation may support some other commands – and may provide other commands for user convenience
- SQL has undergone several standardization efforts: SQL-1, SQL-2, SQL-86, SQL-3, SQL-92, SQL-99; different DBMS have adhered to different standards, but often with extensions

Data Manipulation Language (DML)

```
SELECT  [DISTINCT] ColumnList
FROM TableList
[ WHERE  Condition ]
[ GROUP BY ColumnList
[HAVING condition ]]
[ ORDER BY  ColumnList ]
```

I. Basic Data Retrieval (SELECT)

Find the identifier of each project on which some employee has spent 10 or more hours

```
SELECT  Pno
FROM  Works_On
WHERE  Hours >= 10.0;
```

```
SELECT DISTINCT Pno
FROM Works_On
WHERE  Hours > 10.0;
```

Find the identifier and name of female employees living in Houston & earning more than 30000.

```
SELECT Ssn, Lname, Fname
FROM Employee
WHERE  Sex = 'F'
    AND Salary > 30000
    AND Address LIKE '%Houston%'
ORDER BY Lname, Fname;
```

```
SELECT Ssn, Lname, Fname
FROM Employee
WHERE  Sex = 'F'
    AND Salary > 30000
    AND Address LIKE '%Houston%'
ORDER BY Lname DESC, Fname DESC;
```

Note:

The primary issue here is how to specify logical/Boolean conditions [in the WHERE clause] that identify what data is required by a user:

ColumnName *operator* ColumnName

ComumnName *operator* Value

where the Boolean operator falls into one of the categories below:

Comparison Operator:

= <> (!=) > < >= <=

(NOT = NOT <> NOT < NOT > NOT >= NOT <=)

Set Inclusion Operator:

IN (NOT IN)

Subrange Operator:

BETWEEN (NOT BETWEEN)

Substring Operator:

LIKE (NOT LIKE)

Null Test Operator:

IS NULL (IS NOT NULL)

Boolean conditions can be negated by means of the NOT operator and can be combined by means of the AND and OR operators. As with general programming languages, NOT takes precedence over AND which takes precedence over OR. This precedence of operators can be over-ridden by means of parentheses.

Find the name of employees earning between $20,000 and $30,000

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE  Salary >=  20000
   AND  Salary <=  30000;
```

```
SELECT  Lname, Fname
FROM EMPLOYEE
WHERE  Salary BETWEEN 20000 AND 30000;
```

Find the name of male employees aged between 21 and 25

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Sex = 'M'
   AND  DOB >= '1981-01-20'
   AND  DOB <=  '1986-01-19';
```

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Sex = 'M'
    AND  DOB BETWEEN '1981-01-20' AND '1986-01-19';
```

Find the identifier and name of employees earning less than $25,000 who work in departments 4 or 5

```
SELECT  Ssn, Lname, Fname
FROM  EMPLOYEE
WHERE  Salary < 25000
    AND ( Dno = 4
     OR    Dno = 5 )
```

```
SELECT  Ssn, Lname, Fname
FROM  EMPLOYEE
WHERE  Salary < 25000
    AND  Dno  IN (4, 5)
```

Find the identifier and name of employees earning less than $25,000 who do not work in department nos. 4 or 5

```
SELECT Ssn, Lname, Fname
FROM  Employee
WHERE  Salary < 25000
 AND  Dno != 4
 AND  Dno != 5
```

```
SELECT  Ssn, Lname, Fname
FROM  Employee
WHERE  Salary < 25000
    AND  Dno  NOT IN (4, 5)
```

```
SELECT  Ssn, Lname, Fname
FROM  Employee
WHERE  Salary < 25000
    AND NOT  ( Dno = 4
            OR      Dno = 5 )
```

Find the name and address of employees of department 4 with no specified supervisor

```
SELECT Lname, Fname, Address
FROM EMPLOYEE
WHERE  Dno = 4
    AND  SuperSsn  IS NULL
ORDER BY  Lname DESC, Fname DESC
```

Note:

Most versions of SQL support *string expressions and functions* for usefulness.

e.g. in Oracle:

```
SELECT  Ssn AS "Staff Id",  Lname + ', ' + Fname AS "Staff Name"
FROM Employee
WHERE  Sex = 'F'
   AND  Dno = 5;
```

or, in MySQL:

```
SELECT  Ssn AS "Staff Id",  CONCAT(Lname, ', ', Fname) AS "Staff Name"
FROM Employee
WHERE  Sex = 'F'
   AND  Dno = 5;
```

Some useful MySQL string functions are:

CONCAT (str1, str2, …):	concatenate strings
LEFT (str, len):	returns leftmost *len* characters from string *str*
LENGTH (str):	returns string length
LOWER (str):	returns lower-case version of *str*
REVERSE (str):	reverse the string *str*
RTRIM (str):	removes trailing blanks
SOUNDEX (str):	returns phonetic sequence of *str*

UPPER (str): returns upper-case version of *str*

Find the identifier and name of every employee named Smith

```
SELECT Ssn, UPPER(Lname), Fname
FROM Employee
WHERE  Lname = 'Smith';
```

```
SELECT Ssn, UPPER(Lname), Fname
FROM Employee
WHERE  UPPER(Lname) = 'SMITH';
```

Find the identifier and name of every employee whose name sounds like Smith [Smith, Smyth, Smythe, Smit, …]

```
SELECT Ssn AS "Employee Id", CONCAT(UPPER(Lname), ', ',  Fname)  AS "Employee Name"
FROM Employee
WHERE  SOUNDEX(Lname) = SOUNDEX('Smith');
```

Note:

Most versions of SQL also support *numeric expressions and functions* for usefulness.

Arithmetic Operators: + - * /

e.g.

```
SELECT  Salary/12
FROM Employee
WHERE  Ssn = '123456789'
```

e.g.

```
SELECT Ssn, Salary * 1.06 AS "Employee Cost"
FROM Employee
```

e.g.

```
SELECT Lname + ', ' + Fname AS "Staff Name", Salary / 12 AS "Monthly Salary"
FROM  Employee
WHERE  Salary / 12 > 2000
```

```
SELECT Lname + ', ' + Fname AS "Staff Name", FLOOR(Salary / 12) AS "Monthly Salary"
FROM  Employee
WHERE  Salary / 12 > 2000
```

Some useful MySQL numeric functions are:

ABS(x):	absolute value of *x*
CEILING(x), FLOOR(x):	smallest/largest integer closest to value *x*
COS(x), SIN(x),TAN(x):	sinusoidal functions of *x*
EXP (x):	exponential of *x*
LOG (x):	logarithm of value *x*

Find the name of male employees aged between 21 and 25 *(again)*

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Sex = 'M'
```

```
    AND  DOB BETWEEN DATE_SUB(CURDATE(), INTERVAL 25 YEAR)
                 AND DATE_SUB(CURDATE(), INTERVAL 21 YEAR);
```

II. Statistical Data Retrieval (Functions, GROUP BY, HAVING)

Find the highest salary of all employees

```
SELECT MAX(Salary)
FROM EMPLOYEE
```

Find the average salary for female employees of department no. 5

```
SELECT AVG(Salary)
FROM EMPLOYEE
WHERE Dno = 5
   AND  Sex = 'F'
```

Find the total number of employees that have worked on project no. 20, together with the total number of hours expended on it

```
SELECT COUNT(*) AS "Employee Count", SUM(Hours) AS "Hours Expended"
FROM WORKS_ON
WHERE Pno = 20
```

Find the total number of employees who have female dependents

```
SELECT  COUNT (DISTINCT  Essn)
FROM  DEPENDENT
WHERE  Sex = 'F'
```

Note:

- the group functions differ from the string and numeric functions seen earlier: the latter apply to a specific value(s) in a specific row of a table; the group functions apply to the entire column(s) of the result table of a query
- either '*' or 'DISTINCT Colname' are the standard arguments to the COUNT function
- the group functions are for computation of basic statistical values; it does not make sense – nor is it legal – to mix normal columns and group values in the same SELECT clause [with one exceptional case, as outlined below]
- the group functions are for computation of basic statistical values; it does not make sense – nor is it legal – to mix normal columns and group values in the same SELECT clause [with one exceptional case, as outlined below

The useful MySQL group functions are:

AVG(*expr*):	average value of expr
COUNT(*):	number of rows in result table
COUNT(DISTINCT *expr*):	number of distinct rows containing expr
MAX(*expr*):	maximum value of expr
MIN(*expr*)	minimum value of expr
SUM (*expr*):	summation of all values of expr

The *expr* value of AVG & SUM must be numeric

Find the highest salary in each department

```
SELECT  Dno, MAX(Salary)
FROM EMPLOYEE
GROUP BY Dno
```

Find the average monthly salary for female employees in each department

```
SELECT  Dno, AVG(Salary / 12)
FROM EMPLOYEE
WHERE  Sex = 'F'
GROUP BY Dno
```

Note:

- this last example is the exceptional case, where normal columns and group functions can be used together in SELECT clauses: the column concerned is the one on which grouping occurs in the GROUP BY clause
- the effect of the GROUP BY clause is to:
  - arrange the table (after restriction as defined by the WHERE clause) into similarity groups of rows that match on the specified column(s);
  - treat each group of rows as though it was a separate table for group function calculation]

Find the highest salary in each department with more than five employees

```
SELECT  Dno, MAX(Salary)
FROM EMPLOYEE
GROUP BY Dno
HAVING COUNT(*) >= 5
```

Find the identity of each department, and its average salary, whose average salary is greater than $35,000

```
SELECT  Dno, AVG(Salary)
FROM  EMPLOYEE
```

```
GROUP BY Dno
HAVING AVG(Salary) > 35000
```

III. MULTIPLE TABLE RETRIEVAL USING SUBQUERIES

Find the name of employees working for the Research Department

```
SELECT  CONCAT (UPPER(Lname), ', ', Fname) AS Name
FROM  Employee
WHERE  Dno  IN
    ( SELECT  Dnumber
      FROM  DEPARTMENT
      WHERE  Dname  =  'Research' )
```

Find the name of male employees working in Houston

```
SELECT  CONCAT (Lname, ', ', Fname) AS Name
FROM  EMPLOYEE
WHERE  Sex = 'M'
    AND Dno  IN
        (  SELECT  Dnumber
           FROM  DEPT_LOCATIONS
           WHERE  Dlocation = 'Houston' )
ORDER BY Name
```

Find the identifier and address of female employee working on either 'ProductX' or 'ProductY' projects

```
SELECT   Ssn , Address
FROM EMPLOYEE
WHERE Sex = 'F'
    AND Ssn  IN
    ( SELECT  Essn
      FROM WORKS_FOR
      WHERE  Pno  IN
          ( SELECT  Pnumber
            FROM  PROJECT
            WHERE  Pname IN ('ProductX', 'ProductY'  ) )
```

Find the name(s) of the highest earner / highest female earner

```
SELECT  Lname, Fname
FROM  EMPLOYEE
WHERE  Salary =
    ( SELECT  MAX (Salary)
      FROM  EMPLOYEE  )
```

```
SELECT  Lname, Fname
FROM  EMPLOYEE
WHERE  Salary  >=ALL
        ( SELECT  Salary
          FROM EMPLOYEE  )
```

```
SELECT  Lname, Fname
FROM  EMPLOYEE
WHERE  Sex  =  'F'
   AND  Salary  =
       ( SELECT  MAX (Salary)
         FROM  EMPLOYEE
         WHERE  Sex  =  'F' )
```

```
SELECT  Lname, Fname
FROM  EMPLOYEE
WHERE  Sex  =  'F'
    AND  Salary  >=ALL
       ( SELECT  Salary
         FROM  EMPLOYEE
         WHERE  Sex  =  'F'  )
```

Interblock Connectives

An interblock connective is an operator in the WHERE clause that expresses a condition between a column value and the result returned by a subquery. In the examples above, we used IN, = and >=ALL as interblock connectives. A full list is given below:

Comparison Operators:

= != > < >= <=

Set Comparison Operators:

=ANY <>ANY >ANY <ANY >=ANY <=ANY

=ALL <>ALL >ALL <ALL >=ALL <=ALL

Set Inclusion Operator:

IN (NOT IN)

Existential Quantifier:

EXISTS (NOT EXISTS)

Notes:

1. SQL is said to be block structured in that a query can be organised into blocks (or levels, or subqueries). Each level can be an arbitrary query in its own right and can use any of the features of the SQL language, with three exceptions: the SELECT DISTINCT option, the use of the ORDER BY clause and the renaming of retrieved columns are only permitted at the outer level. Note that all references to columns are local to the current block's FROM clause.

2. All subqueries, except for synchronised subqueries to be seen later, evaluate bottom-up, i.e., each individual level executes to completion and returns a result to the next level up, where it can be used in testing a condition.

3. The comparison operators are valid only if the subquery returns a single row. Otherwise, an error will occur. The set comparison operators should be used when a set of values are expected back from the subquery.

4. When using the set comparison operators, 'ANY' corresponds to common usage of the word some; similarly, 'ALL' would generally be expressed in English as every. Thus, in the last example above, we are searching for the EMPLOYEE member whose salary is greater than or equal to every salary value in existence; obviously, therefore, it must be the largest such value.

5. =Any and IN are equivalent. Similarly, <>ALL is equivalent to NOT IN. =ALL is redundantly false and should not be used; the same goes for <>ANY, which is redundantly true. Thus, the two queries shown below are of no real use.

6. The existential quantifier is used only with a particular type of subquery – synchronised subqueries – which we shall see later.

Redundant Queries:

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Ssn  =ALL
     ( SELECT  Essn
       FROM  WORKS_ON
       WHERE  Hours >= 10.0  )
```

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Ssn  <>ANY
   ( SELECT  Essn
     FROM  WORKS_ON
     WHERE  Hours >= 10.0  )
```

Find the name & address of the lowest-paid employee living in Houston

```
SELECT  Lname, Fname, Address
FROM  EMPLOYEE
WHERE  Address LIKE '%Houston%'
   AND  Salary  =
     ( SELECT  MIN (Salary)
       FROM  EMPLOYEE
       WHERE  Address LIKE '%Houston%'  )
```

or

```
SELECT  Lname, Fname, Address
FROM  EMPLOYEE
WHERE  Address LIKE '%Houston%'
   AND  Salary  <= ALL
     ( SELECT  Salary
       FROM  EMPLOYEE
       WHERE  Address LIKE '%Houston%'  )
```

Find the identifier and name of projects in Bellaire that were never worked on

```
SELECT  Pno, Pname
FROM  PROJECT
WHERE  Plocation  = 'Bellaire'
   AND  Pnumber  NOT IN
    (  SELECT  Pno
       FROM  WORKS_ON  )
```

Find the identifier, name and location of the project(s) which has had the most people working on it

```
SELECT  Pumber, Pname, Plocation
FROM  PROJECT
WHERE  Pnumber  IN
     (   SELECT  Pno
         FROM  WORKS_ON
         GROUP BY  Pno
         HAVING COUNT (*)  =  MAX (COUNT (*) ) )
```

Note:

This last query, while adhering to the SQL standard, will generate an error in MySQL. This is because MySQL does not support embedded function calls – the calculation of a function of a function.

Can you figure out how to rewrite the query so that it does work in MySQL? Hint: You will need one additional level subquery.

Find the name of employees who have the same department and salary as 123456789

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Ssn <> '123456789'
     AND  (Dno, Salary ) =
        (  SELECT  Dno, Salary
           FROM   EMPLOYEE
           WHERE  Sno = '123456789'  )
```

Note:

This query will also generate an error in MySQL, because that DBMS only permits a subquery to return a single column. It could be rewritten using independent subqueries as shown below:

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Ssn <> '123456789'
     AND  Dno   =
          (  SELECT  Dno
             FROM   EMPLOYEE
             WHERE  Ssn = '123456789'  )
     AND  Salary  =
          (  SELECT  Salary
             FROM   EMPLOYEE
             WHERE  Ssn = '123456789'  )
```

However, this technique has its limits. The following query – which purports to find the name of employees who have the same department and salary as John Smith – would produce incorrect results if more than one John Smith existed. Can you see why?

```
SELECT  Lname + ', ' + Fname
FROM  EMPLOYEE
WHERE  ( Fname <> 'John'  OR  Lname <> 'Smith' )
     AND  Dno  IN
        (  SELECT  Dno
           FROM  EMPLOYEE
           WHERE  Fname  =  'John'
                AND  Lname  =  'Smith'  )
```

```
    AND  Salary  IN
       (  SELECT   Salary
          FROM   EMPLOYEE
          WHERE  Fname  =  'John'
              AND  Lname  =  'Smith'  )
```

IV. MULTIPLE TABLE RETRIEVAL USING JOINS

Find the name of employees working for the Research Department

```
SELECT  CONCAT (UPPER(Lname), ', ', Fname) AS Name
FROM  EMPLOYEE, DEPARTMENT
WHERE  Dno  =  Dnumber
    AND  Dname  =  'Research'
ORDER BY Name;
```

In MySQL, the following is also allowed:

```
…
FROM  EMPLOYEE  JOIN  DEPARTMENT
…
```

Find the name of male employees working in Houston

```
SELECT  CONCAT (Lname, ', ', Fname) AS Name
FROM  EMPLOYEE, DEPT_LOCATIONS
WHERE  Sex = 'M'
    AND Dno  =  Dnumber
   AND  Dlocation = 'Houston'
ORDER BY Name;
```

Find the identifier and address of female employee working on either 'ProductX' or 'ProductY' projects

```
SELECT   Ssn , Address
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE Ssn  =  Essn
    AND  Pno  =  Pnumber
    AND Pname IN ('ProductX', 'ProductY'  )
    AND  Sex = 'F';
```

or

```
SELECT   Ssn , Address
FROM EMPLOYEE
WHERE  Ssn  IN
   (  SELECT  Essn
      FROM WORKS-ON ,  PROJECT
      WHERE  Pno  =  P.number
          AND  Pname IN ('ProductX', 'ProductY'  ) )
   AND  Sex = F;
```

Find the name(s) & departments of the highest earner / highest female earner

```
SELECT  Lname, Fname, Dname
FROM  EMPLOYEE, DEPARTMENT
WHERE  Dno=  Dnumber
    AND  Salary =
    ( SELECT  MAX (Salary)
      FROM  EMPLOYEE  )
```

```
SELECT  Lname, Fname,  Dname
FROM  EMPLOYEE, DEPARTMENT
WHERE  Dno  =  Dnumber
    AND  Sex  =  'F'
   AND  Salary  =
        ( SELECT  MAX (Salary)
          FROM  STAFF
          WHERE  Sex  =  'F' )
```

Find the name of employees working in the same department as Joyce English

```
SELECT  RE.Fname, RE.Lname
FROM   EMPLOYEE  LE,  EMPLOYEE RE
WHERE  LE.Dno  =  RE.Dno
    AND  LE.Ssn !=  RE.Ssn
    AND  LE.Fname  =  'Joyce'
    AND  LE.Lname  =  'English'
```

Find the name of employees working in the same department as Joyce English, but earning more than her

```
SELECT  RE.Fname, RE.Lname
FROM   EMPLOYEE  LE,  EMPLOYEE RE
WHERE  LE.Dno  =  RE.Dno
    AND  LE.Ssn !=  RE.Ssn
    AND  LE.Fname  =  'Joyce'
    AND  LE.Lname  =  'English'
    AND  LE.Salary  <  RE.Salary
```

Note:

The second condition above (LE.Ssn != RE.Ssn) is actually redundant in light of the final condition (LE.Salary < RE.Salary)

Find the name of every Texax-based employeer, together with the name & relationship of any dependents he/she might have

```
SELECT  Fname, Lname, Dependent_Name, Relationship
FROM  EMPLOYEE  LEFT JOIN  DEPENDENT
WHERE  Ssn =  Essn
   AND  Adddress LIKE '%TX'
```

- this requests that a *left outer join* be evaluated [any normally unmatched row from the left table will automatically match a null row appended to the right]
- a right outer join also exists in MySQL

- the syntax user here is MySQL-specific; other implementations of the SQL language adopt different syntax

V. MULTIPLE TABLE RETRIEVAL USING SYNCHRONIZED SUBQUERIES

also known as **Correlated Subqueries**
also known as **Subqueries with Interblock Reference**

Find the name of employees working in the Research department

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  'Research' IN
   ( SELECT  Dname
     FROM  DEPARTMENT
     WHERE  Dnumber  =  Dno  )
```

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  ' Research'  =
      ( SELECT  Dname
        FROM DEPARTMENT
        WHERE Dnumber  =  Dno )
```

Note:
The two above queries, while adhering to the SQL language standard, will not be accepted by MySQL. To legitimize them, we must explicitly alias the outer table, and then use that alias within the subquery:

```
SELECT  Fname, Lname
FROM  EMPLOYEE  E
WHERE  'Research' IN
   ( SELECT  Dname
     FROM  DEPARTMENT
     WHERE  Dnumber  =  E.Dno  )
```

```
SELECT  Fname, Lname
FROM  EMPLOYEE  E
WHERE  ' Research'  =
    ( SELECT  Dname
      FROM DEPARTMENT
       WHERE Dnumber  =  E.Dno )
```

Find the name of male employees working in Houston

```
SELECT  Fname, Lname
FROM  EMPLOYEE
WHERE  Sex  =  'M'
    AND  'Houston'  IN
   ( SELECT  Dlocation
     FROM  DEPT_LOCATIONS
     WHERE  Dnumber  =  Dno  )
ORDER BY Name
```

Find the identifier & name of departments hiring employees resident in Houston

```
SELECT  Dnumber, Dname
FROM  DEPARTMENT  D
WHERE  EXISTS
   ( SELECT  *
     FROM  EMPLOYEE
     WHERE  Address  LIKE  '%Houston%'
         AND  Dno  =  D.Dnumber  )
```

Note:
The EXISTS condition returns a *true* value if the subquery returns any data; otherwise it returns *false*. Correspondingly, NOT EXISTS returns *true* when the subquery returns nothing.

Find the identifier & name of departments hiring no employees resident in Houston

```
SELECT  Dnumber, Dname
FROM  DEPARTMENT  D
WHERE  NOT  EXISTS
   (  SELECT  *
      FROM  EMPLOYEE
      WHERE  Address  LIKE  '%Houston%'
          AND  Dno  =  D.Dnumber  )
```

Find the name of all employees working on project no. 20

```
SELECT  Fname, Lname
FROM  EMPLOYEE  E
WHERE  EXISTS
   (  SELECT  *
      FROM  WORKS_ON
      WHERE  Essn  =  E.Ssn
          AND  Pno = 20  )
```

Find the name of staff handling working on no projects

```
SELECT  Fname, Lname
FROM  EMPLOYEE  E
WHERE  NOT EXISTS
   (  SELECT  *
      FROM  WORKS_ON
      WHERE  Essn  =  E.Ssn  )
```

VI. DATA INSERTION, REMOVAL & MODIFICATION

Note:
Unlike retrieval (SELECT statement), the other DML statements (INSERT, DELETE & UPDATE) can potentially leave the database in an inconsistent state. For example, inserting a row into the EMPLOYEE table with a Dno value of 50 is in conflict with the fact that no DEPARTMENT row with Dnumber 50 exists. Similar problems can arise due to deletion or modification of values that implicitly link tables together. Such inconsistencies are termed *referential integrity violation*s.

Database administrators might address this issue in one of three ways:

(i) Do Nothing [or almost nothing].
Database modification might be confined to a small number of highly trained individuals, who are then trusted to do things properly.
Advantage: Easy to do.
Disadvantage: Corruption of the database is inevitable.

(ii) Specify rules that the DBMS might apply to modification commands.
DBMS supports *constraint rules* that can prevent data corruption.
Advantage: Automated consistency checks are possible.

Disadvantage: Will slow down evaluation of SQL data modification commands; handling of problematic commands might not be straightforward; reasons for command rejection might not be obvious to users.

(iii) Perform data modification under program control.
Do not directly use SQL commands for database modification; instead, embed the commands within a programming language (e.g. PHP) that also carries out sanity checks on what the commands are attempting to do.
Advantage: Can apply sophisticated command validation, standards application, etc.
Disadvantage: Requires the writing of programs to handle multiple common data modification processes.

Record details of a new employee.

```
INSERT  INTO EMPLOYEE
VALUES  ( 'Joan', 'J', 'McGregor', '234765980', '1978-01-01', '81 Beech, Austin, TX', 'F', 75000,
NULL, 4);
```

Update details of an employee.

```
UPDATE  EMPLOYEE
SET  Address  =  '18 Maple, Austin, TX',
       Salary = 50000
WHERE  Ssn = '123456789';
```

Increment the salary of employees of the Research department by 10%

```
UPDATE  EMPLOYEE
SET  Salary  =  Salary * 1.1
WHERE  Dno  IN
  (  SELECT  Dnumber
     FROM  DEPARTMENT
     WHERE  Dname  =  'Research'  );
```

```
UPDATE  EMPLOYEE
SET  Salary = Salary * 1.1
FROM  EMPLOYEE, DEPARTMENT
WHERE  Dno = Dnumber
   AND  Dname = 'Research;
```

Remove information on Employee Jennifer Wallace.

```
DELETE
FROM  EMPLOYEE
WHERE  FName = 'Jennifer'
    AND  LName = 'Wallace';
```

Remove information on all dependents of Franklin Wong.

```
DELETE
FROM  DEPENDENT
WHERE  Essn  IN
  (  SELECT  Ssn
     FROM  EMPLOYEE
     WHERE  FName  =  'Franklin'
        AND  LName  =  'Wong'  );
```

```
DELETE  DEPENDENT
FROM EMPLOYEE, DEPENDENT
WHERE  Ssn = Essn
   AND FName  =  'Franklin'
   AND LName  =  'Wong';
```

Note:

We can combine INSERT with SELECT to populate one table with data extracted from another. This is a useful feature for data extraction/restructuring.

e.g. assume we have a newly-created (empty) table:

HOUSTON_STAFF (StaffNo, Name, Dept, Salary)

We could populate this table using existing contents of the EMPLOYEE table:

```
INSERT INTO HOUSTON_STAFF
SELECT  Ssn, CONCAT (LName, ‘, ‘, Fname, ‘ ‘, Minit), Dno, Salary
FROM  EMPLOYEE
WHERE Address LIKE ‘%Houston%’;
```

Data Definition Language (DDL)

CREATE / DROP DATABASE

```
CREATE DATABASE companydb;

CREATE DATABASE IF NOT EXISTS mydatabase;

CREATE DATABASE Dreamhome
ON D:/dbm2                              { Oracle }
INITIAL=4, EXTENT=2
LOG ON D:/dblogs;

DROP DATABASE companydb;
```

CREATE / DROP / ALTER TABLE

```
CREATE TABLE PROJECT
    (  PNAME        CHAR(15),
       PNUMBER      INT,
       PLOCATION    VARCHAR(15),
       DNUM         INT  );

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)  );
```

Note:

The last example shows how a table definition requires a *table name*, *attribute names* and attribute *data types*; it can also contain *constraints* of various types, such as *null constraints*, *key/unique* constraints and *foreign/referential* constraints.

Several variations in syntax are possible, as outlined below:

```
PLOCATION       VARCHAR(15)  DEFAULT 'Houston',

CONSTRAINT UNIQUE (PNAME),

CONSTRAINT OneName UNIQUE (PNAME),       { constraint name }
```

```
CREATE TABLE PROJECT
    (  PNAME          VARCHAR(15)  NOT NULL,
       PNUMBER        INT          NOT NULL,
       PLOCATION      VARCHAR(15),
       DNUM  INT                   NOT NULL,
       PRIMARY KEY (PNUMBER),
       CONSTRAINT  OneName  UNIQUE (PNAME),
       CONSTRAINT  Controller  FOREIGN KEY (DNUM) REFERENCES
              DEPARTMENT(DNUMBER) ON DELETE CASCADE );
```

```
CREATE TABLE EMPLOYEE
        ( FNAME             VARCHAR(15)         NOT NULL ,
          MINIT             CHAR ,
          LNAME             VARCHAR(15)         NOT NULL ,
          SSN               CHAR(9)             NOT NULL ,
          BDATE             DATE
          ADDRESS           VARCHAR(30) ,
          SEX               CHAR ,
          SALARY            DECIMAL(10,2) ,
          SUPERSSN          CHAR(9) ,
          DNO               INT                 NOT NULL ,
    PRIMARY KEY (SSN) ,
   FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
    FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE DEPARTMENT
        ( DNAME             VARCHAR(15)         NOT NULL ,
          DNUMBER           INT                 NOT NULL ,
          MGRSSN            CHAR(9)             NOT NULL ,
          MGRSTARTDATE      DATE ,
       PRIMARY KEY (DNUMBER) ,
       UNIQUE (DNAME) ,
       FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) ) ;
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) ) ;
```

Note:

- the effect of the CREATE TABLE command is to store a descriptor of the table in the *database directory* [*system catalog*]; the effect of ALTER TABLE is to modify that descriptor; the effect of DROP TABLE is to remove the descriptor
- primary/unique columns must have NOT NULL constraints
- MySQL: constraint names are ignored (cannot be dropped in ALTER TABLE statements)
- MySQL: a single variable-length column, explicit or implicit, causes all columns to be silently converted to variable-length data types

Data Types in MySQL

Numeric	String	Date/Time
TINYINT	CHAR	DATE ('CCYY-MM-DD')
SMALLINT	VARCHAR	TIME ('hh:mm:ss')
MEDIUMINT	TINYBLOB	DATETIME
INT	BLOB	TIMESTAMP
BIGINT	MEDIUMBLOB	YEAR ('CCYY')
FLOAT	LONGBLOB	
DOUBLE	TINYTEXT	
DECIMAL	TEXT	
	MEDIUMTEXT	
	LONGTEXT	
	ENUM	
	SET	

Note:

- variants of the same basic type (e.g. INT family) differ in their storage requirements – and upper/lower limits (e.g. 1, 2, 3, 4, 8 bytes, with values up to 127, 32767, 8388607, 2147483647, …); [actually, these values are for signed numbers, unsigned (positive) numbers can double in size]
- these types may be parameterized for storage and/or display purposes:
  - INT (3) - displays in column width 3
  - DECIMAL (7,2) - displays in "7.2" format
  - CHAR (15) - storage 15 characters, padded
  - VARCHAR (15) - max storage 15 characters, unpadded
- the TEXT & BLOB families are implicitly of variable length
- observations:
  - variable-length types are designed for saving of storage space (!)
  - TEXT/BLOB types are relatively uncommon, as they don't fit easily into a table structure; they are rarely used as conditions in queries; when retrieved within queries – not necessarily as part of the condition – they can prove inefficient; to overcome this, these types are often extracted into a separate table

```
ALTER TABLE PROJECT
ADD  START_DATE   DATE;

ALTER TABLE EMPLOYEE
ADD  POSITION  CHAR(12)  NOT NULL;          { legal? }

ALTER TABLE  PROJECT
```

```
DROP  PLOCATION;

ALTER TABLE  EMPLOYEE
MODIFY  ADDRESS  CHAR(50);

DROP TABLE DEPT_LOCATIONS;
```

CREATE / DROP INDEX

```
CREATE INDEX EMPIX ON EMPLOYEE(SSN);

CREATE INDEX WEP ON WORKS_ON(ESSN, PNO);

CREATE UNIQUE INDEX DEPTX ON DEPARTMENT(DNUMBER);

DROP INDEX EMPIX;
```

Note:

- an index is a secondary storage item that should speed up queries on the indexed column(s); it is created if certain query patterns are expected; it may, however, slow down inserts, deletes & updates on the table; note that database operations can never subsequently request that an index be used during evaluation – it is the task of the *query optimizer* to use or ignore the presence of an index in generating an evaluation plan
- indexes can also be created within the CREATE TABLE or ALTER TABLE statements; they can be dropped within the ALTER TABLE statements
- MySQL: automatically converts CREATE INDEX & DROP INDEX statements into equivalent ALTER TABLE statements

Data Control Language (DCL)

User Account Management

```
GRANT  CONNECT
TO  Black
IDENTIFIED BY BlkPswXX;


GRANT  RESOURCE
TO white
IDENTIFIED BY SecretPsw;

GRANT  DBA
TO SysMgr
IDENTIFIED BY TopSecrPsw;


GRANT  RESOURCE
TO Black;

REVOKE RESOURCE
FROM Black;
```

Note:

- Creating a new user account requires a username, a password and a privilege level; adding a privilege level to an existing user is also possible
- Privilege levels determine what that user can subsequently do – specifically what SQL commands he/she can legally execute:
    - CONNECT: login to DBMS only
    - RESOURCE: CONNECT + execute DDL statements
    - DBA: RESOURCE + GRANT/REVOKE privilege

Table Access Management

```
GRANT  SELECT
ON  companydb.employee
TO  white;

GRANT SELECT
ON companydb.*
TO Black
WITH GRANT OPTION;
```

```
GRANT  SELECT, INSERT
ON companydb.department
TO gray
WITH GRANT OPTION;

GRANT SELECT, UPDATE (Salary, SuperSSN)
ON companydb.employee
To  Ruby, Green, Black;

GRANT ALL
ON companydb.*
TO violet
IDENTIFIED BY passwdX

GRANT ALL
ON companydb.*
TO silver
WITH GRANT OPTION;


REVOKE SELECT
ON  companydb.department
FROM  brown;

REVOKE  ALL
ON companydb.*
FROM violet;
```

Note:

- the GRANT & REVOKE commands gives/takes permissions on tables/databases for named database users
- the important permissions are:
  - ALTER [table], INDEX (create & drop), SELECT, INSERT, DELETE, UPDATE, ALL
- the ON clause can specify:
  - database.tablename - a specific table
  - database.* - all tables in a specific database
  - *.* - all databases, all tables (global)
- note that SQL (including MySQL implementation) is achieved by limiting the commands available to a user – if a user cannot execute a command, then he/she is limited in their access to the data/database

- note also that the MySQL system comes configured for one user (the SuperUser, or DBA) who has all permissions