Unit-2

- Introduction to SQL
- Intermediate and Advanced SQL

SQL History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

The SQL language has several parts:

- **Data-definition language (DDL):** The SQL DDL provides commands
  - for defining relation schemas,
  - deleting relations, and
  - modifying relation schemas.
- **Data-manipulation language (DML)**: The SQL DML provides the ability
  - to query information from the database and
  - to insert tuples into,
  - delete tuples from,
  - and modify tuples in the database.
- **Integrity**: The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy.
  - Updates that violate integrity constraints are disallowed.

- **View definition:** The SQL DDL includes commands for defining views
- **Transaction control**: SQL includes commands for specifying the beginning and end points of transactions.
- **Embedded SQL and dynamic SQL:** Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java
- **Authorization**: The SQL DDL includes commands for specifying access rights to relations and views.

SQL Data Definition

Allows the specification of not only a set of relations but also information about each relation, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- The set of indices to be maintained for each relations.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Basic Types

- **char(n):** Fixed length character string, with user-specified length *n*.
- **varchar(n):** Variable length character strings, with user-specified maximum length *n*.
- **int:**Integer (a finite subset of the integers that is machine-dependent).
- **smallint:** Small integer (a machine-dependent subset of the integer domain type).
- **number(p,d):** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- **date:** A calendar date containing a (four-digit) year, month, and day of the month.

Create Table

- **Creating a Database**
  - create database database-name;
  - create database stud_db;
- An SQL relation is defined using the **create table** command:
- **create table** *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$));
  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i
  -
- Example: Schema:branch(branch_name:string,branch_city:string, assets:int);
  - **create table** *branch*
    (*branch_name* char(15) **not null,**
    branch_city char(30),
    assets integer);

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- Example: Declare *branch_name* as the primary key for *branch*
  - **create table** *branch*
    (*branch_name* char(15)**,**
    branch_city char(30),
    assets integer,
    primary key (*branch_name*))
- primary key (A_{j1} , A_{j2} , …, A_{jm}): The primary-key specification says that attributes A_{j1} , A_{j2} , …, A_{jm} form the primary key for the relation. The primary-key attributes
- **primary key** declaration on an attribute automatically ensures **not null** in SQL-92 onwards

foreign key (A_{k1} , A_{k2} , …, A_{kn}) references s:

The foreign key specification says that the values of attributes (A_{k1} , A_{k2} , …, A_{kn}) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation s.

- **Drop command**
- To remove a relation from an SQL database, we use the drop table command.
- The drop table command deletes all information about the dropped relation from the database.
  - drop table r;
  - Ex:
    - Drop table student;
- **Truncate command**
- It is used to delete all the tuples from the table.
  - TRUNCATE TABLE table_name;
  - Ex:
    - Truncate table student;

Alter command

- The **alter table** command is used to add attributes to an existing relation:
- **alter table** *r* **add** *A D*
- where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.
  - All tuples in the relation are assigned *null* as the value for the new attribute.
  - Example: alter table Student add(address char(20));
  -
- To change the **data type of a column** in a table, use the following syntax:
  - ALTER TABLE *r* MODIFY COLUMN *A D*;
  - Example: alter table Student modify(address varchar(30));
- To **Rename a column**
- alter table table-name rename old-column-name to column-name;
- alter table Student rename address to Location;
-

- The **alter table** command can also be used to **drop attributes** of a relation:

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*

  Example:alter table Student drop(address);

  Alter table student drop constraint myprimarykey;
- To change the **name of a table** or relation
  - ALTER TABLE table_name RENAME TO new_table_name;
  - Ex: alter table student rename to stud

Adding constraints

alter table table-name add constraint myprimarykey primarykey (col1,col2..);

Schema

student(sid:number,name:string,branch:string,section:string,age:number,cgpa:number(2,1))

Data Manipulation Language(Modification of the Database)

- **insertion**
- To insert data/row into a relation,
  - we either specify a tuple to be inserted

    1)Insert into r values(valu1,value2,.....valuen);

    Ex: insert into student values('561','ABC','CSE','B',20,9.7);

    2)INSERT into table_name(column1,column2,….) values(data1,data2,……);

    Ex: INSERT into Student(sid,name,branch,age,section,cgpa) values('562','Ravi','CSE',20,'B',9.7);
  - or
  - write a query whose result is a set of tuples to be inserted
  - Ex: insert into r select attr1,attr2,attr3 from r2 where condition;

- **UPDATE** command is used to update the value of a specified attribute of a row or rows
  - UPDATE table_name
    set column_name = value
    where condition;
    - EX:
      UPDATE Student
      set name='Abhi',age=17
      where sid='565';

- **Delete** command is used to delete the rows from a table
  - DELETE from table_name
    where condition;
  - Ex:
    - DELETE from Student
    - where sid=103;

Basic Query Structure

SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification

- **target-list** :A list of attributes of relations
- **relation-list** A list of relation names
- **qualification:**in relation-list qualification Comparisons combined using AND, OR, and NOT
  - Attr op const or Attr1 op Attr2,
    - where op is one of relational operator
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.

Examples:

- Display all the student details

  Select *

  from student ;

- Display distinct names of students

  Select distinct name

  from student

- Display B Section students
  - Select *
  - from student
  - where section='B';

- Display htno,name and cgpa of B Section student
  - Select htno,name,age,cgpa
  - from student
  - where section='B';

Ordering the Display of Tuples

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
Order by column_name sort_order;

- *sort_order may be either **desc** or **asc***

- Display the students order by their names
  - Select * from student orderby name;
- Display id and names of students order by their cgpa
- Select sid,name from student orderby cgpa;
- Display B section students order by their cgpa

```
SELECT * FROM student WHERE section='B' order by cgpa
```

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
- The operator "like" uses patterns that are described using two special characters:
- percent (%). The % character matches any substring(%' stands for 0 or more characters)
- underscore (_). The _ character matches any character.(`_' stands for any one character)

- *Find the ages of students whose name begins and ends with B and has at least three characters.*
- SELECT age
  FROM Student
  WHERE name LIKE `B_ %B'

Aggregate Functions

SQL supports five aggregate operations, which can be applied on any column, say A, of a relation:

- COUNT ([DISTINCT] A): The number of (unique) values in the A column.
  - SELECT COUNT (*) FROM students

- SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
  - Select sum(cost) from Products;

- AVG ([DISTINCT] A): The average of all (unique) values in the A column.
  - Ex: Find the average age of all students
  - SELECT AVG (age) FROM student

- MAX (A): The maximum value in the A column.
  - Find the maximum cgpa of student
  - SELECT MAX(cgpa) FROM student

- MIN (A): The minimum value in the A column.
  - Find the maximum cgpa of student
  - SELECT MIN(cgpa) FROM student

Sailors(sid: integer, sname: string, rating: integer, age: real)

Boats(bid: integer, bname: string, color: string)

Reserves(sid: integer, bid: integer, day: date)

```
CREATE TABLE sailors ( sid integer, sname varchar(32), rating integer, age real, PRIMARY KEY (sid) );

CREATE TABLE boats( bid integer, bname string, color string, PRIMARY
KEY (bid) );

CREATE TABLE reserves ( sid integer , bid integer , day date,
 PRIMARY KEY (sid, bid, day),
FOREIGN KEY (sid) REFERENCES sailors(sid),
FOREIGN KEY (bid) REFERENCES boats(bid) );
```

Sailors

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

- *Find the names and ages of all sailors.*

SELECT DISTINCT S.sname, S.age FROM Sailors S

With DISTINCT

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

Without DISTINCT

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

- Find all sailors with a rating above 7.

```
SELECT S.sid, S.sname, S.rating, S.age
FROM Sailors AS S
WHERE S.rating > 7
```

- Find the names of sailors who have reserved boat number 103

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid=103
    OR
SELECT sname
FROM Sailors, Reserves
WHERE Sailors.sid=Reserves.sid AND bid=103
```

Sailors S

sid	*sname*	*rating*	*age*
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Reserves R

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

S× R

sid	*sname*	*rating*	*age*	*sid*	*bid*	*day*
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Result

sname
rusty

- Find the sids of sailors who have reserved a red boat

```
Select  R.sid
FROM Boats B, Reserves R
WHERE B.bid = R.bid AND B.color = `red'
```

- *Find the names of sailors who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
```

- *Find the colors of boats reserved by Lubber*

```
SELECT B.color
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND S.sname = `Lubber'
```

- *Find the names of sailors who have reserved at least one boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
```

Expressions and Strings in the SELECT Command

- Each item in a **select-list** can be of the form
  - *expression* AS *column name*,
  - where *expression* is any arithmetic or string expression over column names and constants

- *Compute the increments ratings of persons who have sailed two different boats on the same day.*

```
SELECT S.sname, S.rating+1 AS rating
FROM Sailors S, Reserves R1,Reserves R2
Where S.sid=R1.sid AND S.sid=R2.sid AND R1.day=R2.day AND R1.bid<>R2.bid;
```

- *Find the ages of sailors whose name begins and ends with B and has at least three characters.*

- SELECT S.age
- FROM Sailors S
- WHERE S.sname LIKE `B_ %B'

- *Find the names of sailors who have reserved a red or a green boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
AND (B.color = `red' OR B.color = `green')
```

- *Find the names of sailors who have reserved both a red and a green boat.*

```
SELECT S.sname
FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2
WHERE S.sid = R1.sid AND R1.bid = B1.bid
AND S.sid = R2.sid AND R2.bid = B2.bid
AND B1.color=`red' AND B2.color = `green'
```

Set Operations(UNION, INTERSECT, AND EXCEPT)

- SQL provides three set-manipulation constructs that extend the basic query form presented earlier

  UNION
  INTERSECT
  EXCEPT

- *Find the names of sailors who have reserved a red or a green boat*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
UNION
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green‘
```

- *Find the names of sailors who have reserved both a red and a green boat*

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = `red'
INTERSECT
SELECT S2.sname
FROM Sailors S2, Boats B2, Reserves R2
WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = `green'
```

- *Find the sids of all sailors who have reserved red boats but not green boats.*

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND B.color = `red'
EXCEPT
SELECT R2.sid
FROM Boats B2, Reserves R2
WHERE R2.bid = B2.bid AND B2.color = `green'
```

- *Find all sids of sailors who have a rating of 10 or have reserved boat 104*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating = 10
UNION
SELECT R.sid
FROM Reserves R
WHERE R.bid = 104
```

- SQL also provides other set operations:
  - IN -to check if an element is in a given set
  - op ANY, op ALL -to compare a value with the elements in a given set, using comparison operator op
  - EXISTS - to check if a set is empty

NESTED QUERIES

- A **nested query is a query** that has another query embedded within it
- the embedded query is called a **sub query.**
- A sub query typically appears within the WHERE clause of a query
- Sub queries can sometimes appear in the FROM clause or the HAVING clause

- *Find the names of sailors who have reserved boat 103.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid = 103 )
```

- *Find the names of sailors who have reserved a red boat.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM Reserves R
                 WHERE R.bid IN ( SELECT B.bid
                                  FROM Boats B
                                  WHERE B.color = `red' ))
```

- *Find the names of sailors who have not reserved a red boat.*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN ( SELECT R.sid
                     FROM Reserves R
                     WHERE R.bid IN ( SELECT B.bid
                                      FROM Boats B
                                      WHERE B.color = `red' ))
```

Correlated Nested Queries

- In the nested queries that we have seen thus far, the inner sub query has been completely independent of the outer query
- In general the inner sub query could depend on the row that is currently being examined in the outer query

- *Find the names of sailors who have reserved boat number 103*

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS ( SELECT *
               FROM Reserves R
               WHERE R.bid = 103
               AND R.sid = S.sid )
```

- The EXISTS operator is another set comparison operator, such as IN
  - It allows us to test whether a set is nonempty
  - for each Sailor row *S, we test whether the set* of Reserves rows *R such that R.bid = 103 AND S.sid = R.sid is nonempty.*
  - If so, sailor *S has reserved boat 103, and we retrieve the name*

- The sub query clearly depends on the current row *S and must be re-evaluated for each row in Sailors.*
- *The occurrence* of *S in the sub query (in the form of the literal S.sid) is called a correlation, and such* queries are called *correlated queries*
- using NOT EXISTS instead of EXISTS, we can compute the names of sailors who have not reserved a red boat

Set-Comparison Operators

- SQL also supports op ANY and op ALL,
  - where op is one of the arithmetic comparison operators
  - <;<=;=; <>;>=;>
  - *SOME is also available,* but it is just a synonym for ANY

- *Find sailors whose rating is better than some sailor called Horatio*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                       FROM Sailors S2
                       WHERE S2.sname = `Horatio' );
```

- *Find sailors whose rating is better than every sailor called Horatio.*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL ( SELECT S2.rating
                       FROM Sailors S2
                       WHERE S2.sname = `Horatio' );
```

- *Find the sailors with the highest rating.*

```
SELECT S.sid
FROM Sailors S
WHERE S.rating >= ALL ( SELECT S2.rating
                        FROM Sailors S2 )
```

- Find the names of sailors who have reserved both a red boat and a green boat

```
Select s.sname
from sailors s
Where s.sid IN((select R.sid
                from Boats B, Reserves R
                where R.bid=B.bid AND B.color='red')
               INTERSECT
               (select R.sid
                from Boats B2, Reserves R2
                where R2.bid=B2.bid AND B2.color='green'));
```

- Find the Names of sailors who have reserved all boats

```
Select s.sname
from sailors s
Where NOT EXISTS(select B.bid
                 from Boats B
                 where NOT EXISTS( select R.bid
                                   from Reserves R
                                   where R.bid=B.bid
                                   AND   R.sid=S.sid));
```

- *Count the number of sailors.*
  - SELECT COUNT (*)
  - FROM Sailors S;

- *Count the number of different sailor names.*
  - SELECT COUNT (DISTINCT S.sname)
  - FROM Sailors S

• *Find the name and age of the oldest sailor.*

- SELECT S.sname, MAX (S.age)
- FROM Sailors S

• if the SELECT clause uses an aggregate operation, then it must use *only aggregate operations unless* the query contains a GROUP BY clause

- SELECT S.sname, S.age
- FROM Sailors S
- WHERE (SELECT MAX (S2.age)
- FROM Sailors S2) = S.age

• *Find the names of sailors who are older than the oldest sailor with a rating of 10.*

- SELECT S.sname
- FROM Sailors S
- WHERE S.age > *(SELECT MAX (S2.age)*
  FROM Sailors S2
  WHERE S2.rating = 10)

OR

- SELECT S.sname
- FROM Sailors S
- WHERE S.age > *ALL (SELECT S2.age*
  FROM Sailors S2
  WHERE S2.rating = 10)

The GROUP BY and HAVING Clauses

• Often we want to apply aggregate operations to each of a number of groups of rows in a relation,

- where the number of groups depends on the relation instance

- SELECT [DISTINCT] select-list
- FROM from-list
- WHERE qualification
- GROUP BY grouping-list
- HAVING group-qualification

- The select-list in the SELECT clause consists of
  - (1) a list of column names and
  - (2) a list of terms having the form aggop (*column-name) AS new-name.*
- Every column that appears in (1) must also appear in grouping-list.
  - The reason is that each row in the result of the query corresponds to one *group, which is a* collection of rows that agree on the values of columns in grouping-list.
  - If a column appears in list (1), but not in grouping-list, it is not clear what value should be assigned to it in an answer row.

- The expressions appearing in the group-qualification in the HAVING clause must have a *single value per group.*
  - a column appearing in the group-qualification must appear as the argument to an aggregation operator, or it must also appear in grouping-list.
- If the GROUP BY clause is omitted, the entire table is regarded as a single group.

- Find the age of the youngest sailor for each rating level.
- SELECT S.rating, MIN (S.age)
- FROM Sailors S
- GROUP BY S.rating

- *Find the age of the youngest sailor who is eligible to vote for each rating level with at least two such sailors.*
  - SELECT S.rating, MIN (S.age) AS minage
  - FROM Sailors S
  - WHERE S.age >= *18*
  - GROUP BY S.rating
  - HAVING COUNT (*) > *1*

• *For each red boat, find the number of reservations for this boat.*
  - SELECT B.bid, COUNT (*) AS sailorcount
  - FROM Boats B, Reserves R
  - WHERE R.bid = B.bid
  - GROUP BY B.bid
  - HAVING B.color = `red'

• *Find the average age of sailors for each rating level that has at least two sailors.*
  - SELECT S.rating, AVG (S.age) AS avgage
  - FROM Sailors S
  - GROUP BY S.rating
  - HAVING COUNT (*) > *1*

• *Find those ratings for which the average age of sailors is the minimum over all ratings.*

```
SELECT Temp.rating, Temp.avgage
FROM ( SELECT S.rating, AVG (S.age) AS avgage,
FROM Sailors S
GROUP BY S.rating) AS Temp
WHERE Temp.avgage = ( SELECT MIN (Temp.avgage) FROM Temp )
```

NULL VALUES

- SQL provides a special column value called null .
- We use null when the column value is either unknown or inapplicable.
- The presence of null values complicates many issues, and we consider the impact of null values

- SQL also provides a special comparison operator IS NULL to test whether a column value is *null;*
- IS NOT NULL

- Example: select loan_number from loan where amount is null