Basic SQL

Introduction I

- The Structured Query Language (SQL) has several parts:
  1. **Data-definition language (DDL)** provides commands for defining relation schemas, deleting relations, and modifying relation schemas
  2. **Data-manipulation language (DML)** provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
  3. **Integrity** includes commands for specifying integrity constraints that the data stored in the database must satisfy.
     - Updates that violate integrity constraints are disallowed.
  4. **View definition** includes commands for defining views
  5. **Transaction control** includes commands for specifying the beginning and ending of transactions.

Introduction II

6. **Embedded and dynamic SQL** define how SQL statements can be embedded within general-purpose programming languages
7. **Authorization** includes commands for specifying access rights to relations and views.

Basic Types

- **char(n):** A fixed-length character string with user-specified length n. The full form, character, can be used instead.
- **varchar(n):** A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent
- **int:** An integer. The full form, integer, is equivalent
- **smallint:** A small integer
- **numeric(p, d):** A fixed-point number with user-specified precision

  - number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point.
- **float(n):** A floating-point number, with precision of at least n digits

Basic Schema Definition - CREATE TABLE construct I

```
create table r
(A1 D1,
 A2 D2,
 . . . ,
 An Dn,
 <integrity-constraint1>,
 . . . ,
 <integrity-constraintk>);

create table instructor
(ID varchar(5),
 name varchar(20) not null,
 deptname varchar (20),
 primary key (ID),
 foreign key (deptname) references department);
```

Basic Schema Definition - CREATE TABLE construct II

```
create table classroom
(building varchar (15),
roomnumber varchar (7),
capacity numeric (4,0),
primary key (building, roomnumber));

create table department
(deptname varchar (20),
building varchar (15),
budget numeric (12,2) check (budget > 0),
primary key (deptname));
```

Basic Schema Definition - CREATE TABLE construct III

```
create table course
(courseid varchar (8),
title varchar (50),
deptname varchar (20),
credits numeric (2,0) check (credits > 0),
primary key (courseid),
foreign key (deptname) references department
on delete set null);
```

Basic Schema Definition - CREATE TABLE construct IV

```
create table section
(courseid varchar (8),
secid varchar (8),
semester varchar (6) check (semester in
      ('Fall', 'Winter', 'Spring', 'Summer')),
year numeric(4,0) check(year>1701 and year<2100),
building varchar (15),
roomnumber varchar (7),
timeslotid varchar (4),
primary key (courseid, secid, semester, year),
foreign key (courseid) references course
on delete cascade,
foreign key (building, roomnumber) references
                classroom on delete set null);
```

Modification of DB - Insertion

- Add a new tuple to **course** table

```
insert into course
    values('CS-437','DBS','Comp. Sci.', 4);
```

- or equivalently

```
insert into course(courseid, title, deptname,
 credits) values ('CS-437','DBSystems',
                  'Comp. Sci.', 4);
```

- Add a new tuple to student with totcreds set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

Modification of DB - Insertion

- Add a new tuple to **course** table

```
insert into course
    values('CS-437','DBS','Comp. Sci.', 4);
```

- or equivalently

```
insert into course(courseid,title,deptname,
 credits) values ('CS-437','DBSystems',
                                'Comp. Sci.', 4);
```

- Add a new tuple to student with totcreds set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

Modification of DB - Insertion

- Add a new tuple to **course** table

```
insert into course
    values('CS-437','DBS','Comp. Sci.', 4);
```

- or equivalently

```
insert into course(courseid,title,deptname,
 credits) values ('CS-437','DBSystems',
                                'Comp. Sci.', 4);
```

- Add a new tuple to student with totcreds set to null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

Modification of DB - Delete Construct/Drop Table

- Deleting all the contents of the table

```
delete from student;
```

- Deleting a specific content from the table

```
delete from student
where P;
```

```
delete from student
where deptname = 'ICT';
```

- Deleting table

```
drop table student;
```

Modificatin of DB - Updation I

- Annual salary increases are being made, and salaries of all instructors are to be increased by 5 percent

```
update instructor
set salary = salary * 1.05;
```

- If a salary increase is to be paid only to instructors with salary of less than ₹70,000

```
update instructor
set salary = salary * 1.05
where salary < 70000;
```

- Increase salaries of instructors whose salary is over ₹100,000 by 3%, and all others receive a 5% raise

Modificatin of DB - Updation II

```
update instructor
set salary = salary * 1.03
where salary >= 100000;

update instructor
set salary = salary * 1.05
where salary < 100000;
```

- Order is important!

Modificatin of DB - Updation III

- Same query as before but with case statement

```
update instructor
set salary = case
when salary >= 100000 then salary * 1.03
else salary * 1.05
end;
```

Alter table construct I

- Used to add or drop an attribute to/from a table

```
alter table instructor add age int;
```

- Delete the attribute

```
alter table instructor drop age;
```

```
alter table instructor drop column age;
```

Alter table construct II

- Adding a foreign key

```
alter table B
    add foreign key (name) references A;

alter table B
    add constraint fk_name
    foreign key (name) references A(name);

alter table B
    drop constraint fk_name;

create table B(id number primary key,
name varchar(10),
constraint fk_name foreign key (name)...);
```

Alter table construct III

Basic Query Structure

- A typical SQL query has the following form:

```
select A1, A2, ... An
from r1, r2, ... rm
where P;
```

- The result of an SQL query is a relation

SELECT Clause I

- Select clause lists the attributes desired in the result of a query.
- Find the names of all instructors:

```
select name
from instructor;
```

- SQL names are case insensitive
- Find the department names of all instructors,

```
select deptname
from instructor;
```

- SQL allows duplicates in relations as well as in query results

SELECT Clause II

- To force the elimination of duplicates, insert the keyword distinct after select.

Q Find the names of all departments with instructor, and remove duplicates

- The keyword **all** specifies that duplicates not be removed.
- An asterisk in the select clause denotes "all attributes"
- The select clause can contain arithmetic expressions involving the operation, $+$, -, *, and $/$, and operating on constants or attributes of tuples.

Q Return a relation that is the same as the instructor relation, except that the value of the attribute salary is multiplied by 1.1

WHERE Clause

- The where clause specifies conditions that the result must satisfy

Q Find all instructors in Comp. Sci. dept with salary greater than ₹80000

- Comparison results can be combined using the logical connectives **and, or,** and **not**.
- Comparisons can be applied to results of arithmetic expressions

Queries on Multiple Relations I

Q Retrieve the names of all instructors, along with their department names and department building name

- The role of each clause is as follows:
  - The select clause is used to list the attributes desired in the result of a query.
  - The from clause is a list of the relations to be accessed in the evaluation of the query.
  - The where clause is a predicate involving attributes of the relation in the from clause
- Operational order: first **from**, then **where**, and then **select**
- The from clause by itself defines a Cartesian product of the relations listed in the clause.

Queries on Multiple Relations II

```
select *
from instructor, department;
```

Displays the Cartesian product of every tuple in the relations

- Instead, if we say

```
select * from instructor, department
where instructor.deptname=department.deptname;
```

This displays the details of the instructors only once.

Queries on Multiple Relations III

department(deptname, building, budget)
course(courseid, title, deptname, credits)
instructor(ID, name, deptname, salary)
section(courseid, sectionid, sem, year, building, roomnumber, timeslotid)
teaches(ID, courseid, secid, semester, year)

- Q For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught
- Q Find instructor names and course identifiers for instructors in the Computer Science department
- Q Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Natural Join I

- List the names of instructors along with the course ID of the courses that they taught.

```
select name, courseid
from instructor, teaches
where instructor.id=teaches.id;
```

```
select name, courseid
from instructor natural join teaches;
```

Natural Join II

- List the names of instructors along with the titles of courses that they teach

```
select name, title
from instructor natural join teaches
                natural join course;
```

- Is the query correct? Will this work if there is an instructor who teaches a course that belongs to another department?

Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause

Q For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught

```
select name, courseid
from instructor i, teaches t
where i.id=t.id;
```

- Another usage of rename operation is a case where we wish to compare tuples in the same relation.

Q Find the names of all instructors who have a higher salary than atleast one instructor in 'Comp. Sci'.

- Also known as Correlation name, correlation variable, tuple variable, table alias

String Operations I

- Strings are specified in single quotes. Ex: 'Computer'
- 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
  - **underscore (_):** The _ character matches any character
- Find the names of all instructors whose name includes the substring "dar"

```
select name
from instructor
where name like %dar%
```

String Operations II

- Match the string "100 %"

```
like '100\%' escape '\' )
```

- **like** 'ab\%cd%' **escape** '\'
  matches all strings beginning with "ab%cd".
- **like** 'ab\\cd%' **escape** '\'
  matches all strings beginning with "ab\cd".

Ordering the display of Results

- The **order by** clause causes the tuples in the result of a query to appear in sorted order.

Q List all the instructors who work in Physics department in ascending and descending order.

- Specify **desc** for descending order or **asc** for ascending order, for each attribute

Where clause predicate: between-and

Q Find the names of instructors with salary amounts between ₹90,000 and ₹100,000,

A **where** salary **between** 90000 **and** 100000;

Aggregate Functions I

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value
- SQL offers five built-in aggregate functions
  - Average: **avg**
  - Minimum: **min**
  - Maximum: **max**
  - Total: **sum**
  - Count: **count**
- Input to sum and avg must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.

Aggregate Functions II

Q Find the average salary of instructors in the Computer Science department.

Q Find the total number of instructors who teach a course in the Spring 2010 semester

Q Find the number of tuples in the course relation

- There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples - **group by** clause

Aggregate Functions III

Q Find the average salary of instructors in each department

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Q Find the number of instructors in each department who teach a course in the Spring 2010 semester.

Aggregate Functions IV

- Any attribute that is not present in the **group by** clause must appear only inside an aggregate function if it appears in the select clause, otherwise the query is treated as erroneous

```
/* erroneous query*/
select deptname, name, count(distinct id)
from instructor natural join teaches
where semester = 'Spring' and year = 2010
group by deptname;
```

- At times, it is useful to state a condition that applies to groups rather than to tuples - **having** clause

Q Find departments where the average salary of the instructors is more than ₹42,000

Aggregate Functions V

- Any attribute that is present in the **having** clause without being aggregated must appear in the **group by** clause, otherwise the query is treated as erroneous.
- Order of execution: **from, where, group by, having, order by select**

Q For each course section offered in 2019, find the average total credits (totcred) of all students enrolled in the section, if the section had at least 2 students.

Aggregate Functions VI

```
select courseid, semester, year,
                        secid, avg(totcred)
from takes natural join student
where year = 2019
group by courseid, semester, year, secid
having count (ID) >= 2;
```

Aggregation with Null and Boolean Values

- Null values, when they exist, complicate the processing of aggregate operators

```
select sum (salary)
from instructor;
```

- All aggregate functions except count (*) ignore null values in their input collection

Set Operations I

- SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations

Q Find the set of all courses offered in the Fall 2015 semester

```
select courseid
from section
where semester = 'Fall' and year = 2015;
```

Q Find the set of all courses offered in the Spring 2016 semester

```
select courseid
from section
where semester = 'Spring' and year = 2016;
```

Set Operations II

Q Find the set of all courses offered either in Fall 2015 or in Spring 2016, or both.

```
select courseid
from section
where semester='Fall' and year=2015
union
select courseid
from section
where semester='Spring' and year=2016;
```

- The union operation automatically eliminates duplicates, unlike the select clause.
- If the duplicates are needed then **union all** is to be used.

Set Operations III

- Q Find courses that were offered in Fall 2015 as well as in Spring 2016 -(**intersect**)
- Q Find courses that ran in Fall 2015 but not in Spring 2016 - (**except or minus**)

Null Values

- null signifies an unknown value or that a value does not exist
- **and:** The result of true and unknown is unknown,
  false and unknown is false, while
  unknown and unknown is unknown
- **or:** The result of true or unknown is true,
  false or unknown is unknown, while
  unknown or unknown is unknown.
- **not:** The result of not unknown is unknown.

Q Find all instructors who appear in the instructor relation with null values for salary.

- **is null, is not null**

Nested Subquery

- A subquery is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality
- Set membership is checked using **in** and **not in** constructs

Nested Subquery - Set Membership

- The **in** connective tests for set membership, where the set is a collection of values produced by a select clause.
- The **not in** connective tests for the absence of set membership.

Q Find all the courses offered in the both the Fall 2015 and Spring 2016 semesters.

Q Find courses offered in Fall 2015 but not in Spring 2016

Q Find the names of instructors whose names are neither "Mozart" nor "Einstein".

Q Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 110011

Nested Subquery - Set Comparison I

Q Find the names of all instructors whose salary is greater than at least one instructor in the Biology department.

- **"greater than at least one"** is represented in SQL by $>$**some**
- The $>$ **some** comparison in the where clause of the outer select is true if the salary value of the tuple is greater than at least one member of the set of all salary values for instructors in Biology.
- SQL also allows $<$ **some,** $<=$ **some,** $>=$ **some,** $=$ **some**, and $<>$ **some** comparisons.

Q Find the names of all instructors who have a salary value greater than that of each instructor in the Biology department.

- Construct $>$ **all** corresponds to the phrase "greater than all"

Nested Subquery - Set Comparison II

- SQL also allows $<$ **all,** $<=$ **all,** $>=$ **all,** $=$ **all**, and $<>$ **all** comparisons
- $<>$**all** is identical to **not in**, whereas **= all** is not the same as **in**.

Q Find the departments that have the highest average salary

Test for empty relation I

- The **exists** construct returns the value true if the argument subquery is nonempty
- **exists** r $\Leftrightarrow$r $\neq \phi$ and **not exists** r $\Leftrightarrow$r $= \phi$

Q Find all courses offered in both the Fall 2009 semester and in the Spring 2010 semester

```
select courseid
from section S
where semester = 'Fall' and year = 2009
and exists (select *
                from section T
                where semester = 'Spring' and
                year = 2010 and
                        S.courseid= T.courseid );
```

Test for empty relation II

- **Correlated subquery:** A subquery that uses a correlation name from an outer query
- Result of EXISTS is a boolean value True or False
- The EXISTS operator terminates the processing of the subquery once the subquery returns the first row.

```
select *
from customers
where exists (select *
              from order_details
              where customers.customer_id =
              order_details.customer_id);
```

Test for empty relation III

- Will return all records from the customers table where there is at least one record in the order_details table with the matching customer_id

```
select *
from customers
where not exists (select *
                  from order_details
                  where customers.customer_id =
                  order_details.customer_id);
```

- Will return all records from the customers table where there are no records in the order_details table for the given customer_id.
- **not exists** construct simulates the set containment

Test for empty relation IV

IN	Exists
scans all rows returned by the subquery to conclude the result.	terminates the processing of the subquery once the subquery returns the first row
return all rows where the attribute value is present in the subquery	returns true if the subquery returns any rows, otherwise, it returns false

`select * from table_name where id in(subquery);`
 	
`select * from table_name where id = 1 OR id = 2 OR id = 3 OR id = NULL;`
	Exists or Not Exists solely checks the existence of rows in the subquery

Test for empty relation V

Q Find those instructors who do not teach any course.

Q Find all students who have taken all courses offered in the Biology department

```
select id, name
from student s
where not exists ((select courseid
                   from course
                   where deptname='Biology')
                   minus
                  (select courseid
                   from takes t
                   where s.id=t.id));
```

Subqueries in FROM Clause I

- Any select-from-where expression returns a relation as a result and, therefore, can be inserted into another select-from-where anywhere that a relation can appear

Q Find the average instructors' salaries of those departments where the average salary is greater than ₹42,000

Q Find the maximum salary across all departments of the total salary at each department

- **from** clause that is prefixed by the **lateral** keyword to access attributes of preceding tables or subqueries in the **from** clause

Q Print the names of each instructor, along with their salary and the average salary in their department

Subqueries in FROM Clause II

- Oracle, does not support renaming of the result relation in the from clause
- Nested subqueries in the from clause cannot use correlation variables from other relations in the from clause.

WITH Clause I

- The **with** clause provides away of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs

Q Find those departments with the maximum budget

- The **with** clause defines the temporary relation, which is used in the immediately following query
- It also permits a view definition to be used in multiple places within a query.

Q Find all departments where the total salary is greater than the average of the total salary at all departments

WITH Clause II

```
with depttotal(deptname, val) as
(select deptname, sum(salary)
 from instructor
 group by deptname),
depttotavg(val) as
 (select avg(val)
  from depttotal)
select deptname, depttotal.val
from depttotal, depttotavg
where depttotal.val>=depttotavg.val;
```

Scalar Subqueries I

- The subquery returns only one tuple containing a single attribute; such subqueries are called **scalar subqueries**
- Scalar subquery is one which is used where a single value is expected

Q List all departments along with the number of instructors in each department

```
select deptname, (select count(*)
                  from instructor
                  where department.deptname =
                        instructor.deptname)
                  as numinstructors
from department;
```

Scalar Subqueries II

- Scalar subqueries can occur in select, where, and having clauses
- Scalar subqueries may also be defined without aggregates
- If the result has more than one tuple when the subquery is executed, a run-time
- Technically the type of a scalar subquery result is still a relation, even if it contains a single tuple error occurs

Modification of Database - Deletion

- Q Delete all tuples in the instructor relation pertaining to instructors in the Finance department
- Q Delete all instructors with a salary between ₹13,000 and ₹15,000
- Q Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building.
- Q Delete the records of all instructors with salary below the average at the university
- Performing all the tests before performing any deletion is important

Modification of Database - Insertion and Updation I

Q Make each student in the Music department who has earned more than 144 credit hours, an instructor in the Music department, with a salary of ₹18,000.

- It is possible for inserted tuples to be given values on only some attributes of the schema.

Q Update totcred attribute of each student tuple to the sum of the credits of courses successfully completed by the student. Assume that a course is successfully completed if the student has a grade that is not 'F' or null

Modification of Database - Insertion and Updation II

```
update student S
set totcred = (select sum(credits)
               from takes natural join course
               where S.ID= takes.ID
               and takes.grade <> 'F' and
               takes.grade is not null);
```

1. Find the titles of courses in the Comp. Sci. department that have 3 credits.
2. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result
3. Find the highest salary of any instructor
4. Find all instructors earning the highest salary (there may be more than one with the same salary).
5. Find the enrollment of each section that was offered in Autumn 2009
6. Find the maximum enrollment, across all sections, in Autumn 2009
7. Find the sections that had the maximum enrollment in Autumn 2009

8. Increase the salary of each instructor in the Comp. Sci. department by 10%.

9. Delete all courses that have never been offered (that is, do not occur in the section relation

10. Insert every student whose total credit attribute is greater than 100 as an instructor in the same department, with a salary of ₹10,000.

11. Find the names of all studentswho have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result

12. Find the IDs and names of all students who have not taken any course offering before Spring 2009

13. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor

14. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query