**PDF 1**

1. create table r1(dep varchar2(20), depno numeric(20), code varchar2(5),

budg numeric(20));

1. alter table r1 add constraint pk primary key (code);
2. drop table A;
3. delete from dept;
4. alter table course add D DATE;
5. alter table course drop column D;

**PDF 2**

1. select name from instructor;
2. select all semester from courses;
3. select course\_id, course\_name, year\*2 from courses;
4. select \* from courses;

COURSE\_ID COURSE\_NAM INSTRUCTOR SEMESTER YEAR

---------- ---------- ---------- ---------- ----------

101 DBMS M.Smith Fall 2009

102 OS S.Johnson Fall 2009

103 CN P.Jason Fall 2009

104 DSA J.Kohli Fall 2009

101 DBMS M.Smith Spring 2010

103 CN P.Jason Spring 2010

105 AI A.Davis Spring 2010

106 SWE Q.Aavinder Spring 2010

8 rows selected.

1. select course\_name from courses where instructor='M.Smith' and year>2009;

COURSE\_NAM

----------

DBMS

1. create table C(err varchar2(10) primary key, y numeric(10), constraint fk foreign key(err) references I(err)); //same type
2. CREATE TABLE my\_table (

id NUMBER PRIMARY KEY,

name VARCHAR2(50),

created\_date DATE

);

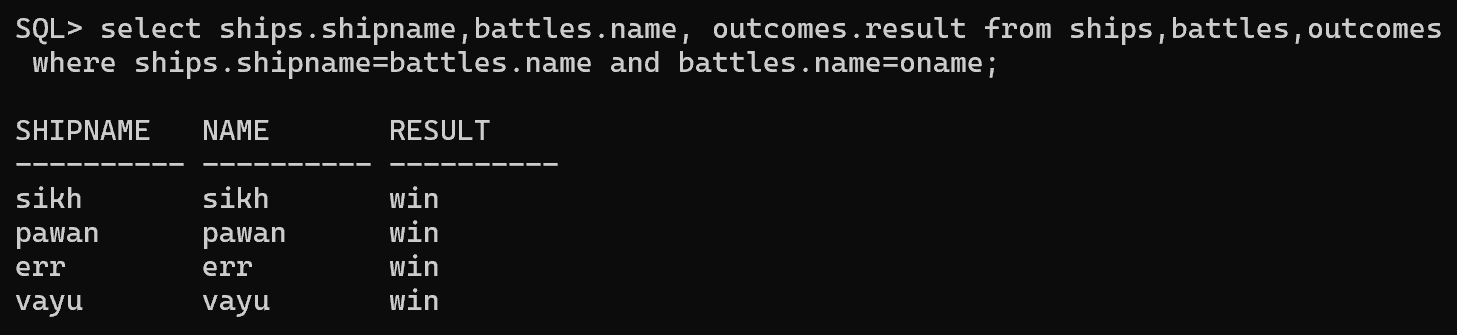
1. INSERT INTO my\_table (id, name, created\_date)

VALUES (1, 'Example', TO\_DATE('2024-11-21', 'YYYY-MM-DD'));

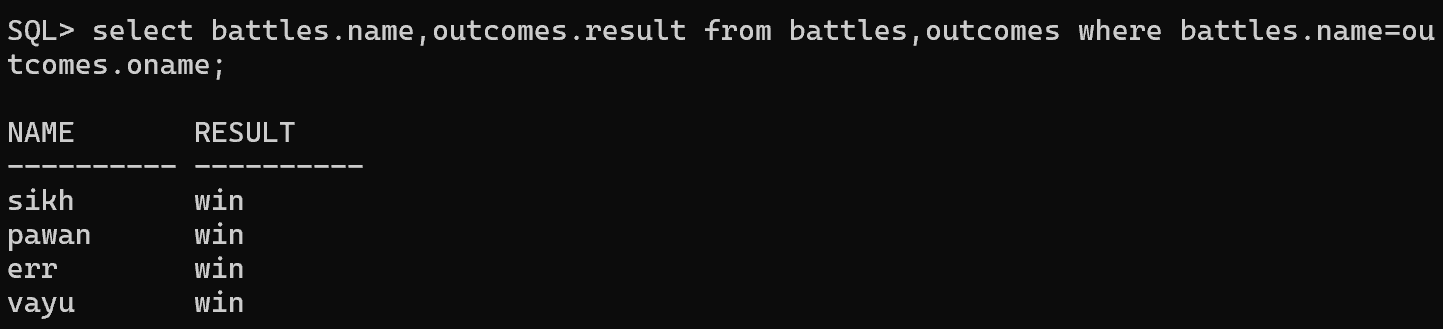
1. ALTER TABLE table\_name DROP PRIMARY KEY;
2. ALTER TABLE table\_name ADD CONSTRAINT constraint\_name PRIMARY KEY (column\_name);
3. alter table ships drop primary key;
4. alter table ships add constraint pks primary key(shipname);
5. create table ships(shipname varchar2(10) primary key, id number(10),year varchar2(10) not null);

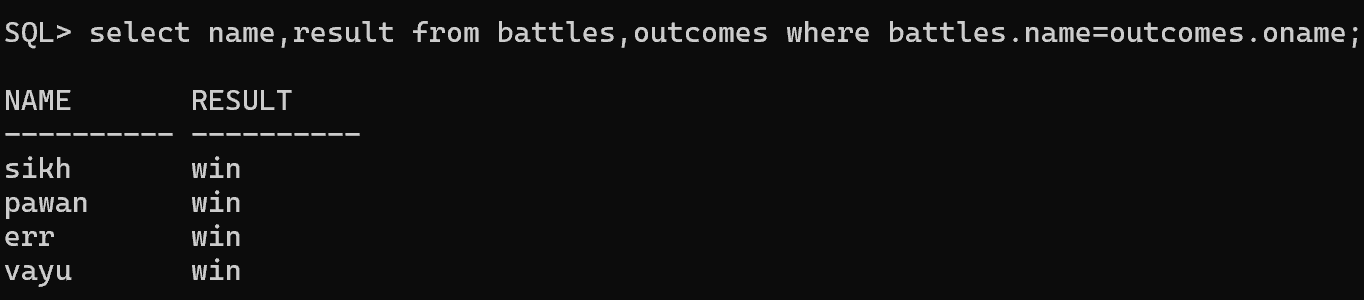
create table battles(name varchar2(10) primary key, dat DATE not null, shipname varchar2(10) not null, nships number(10) not null, constraint fck foreign key(name) references ships(shipname));

create table outcomes(oname varchar2(10) primary key, result varchar(10) not null, constraint fkc foreign key(oname) references battles(name));



1. create table bla(i varchar2(10) primary key, u numeric(10), constraint fak foreign key(i) references tabla(i) );





select avg(sal) from A where id='20';

select avg(sal) as salary from A where id='20'

select count (distinct ID) from teaches where semester = 'Spring' and year = 2018;

1. select count(\*) from A;
2. SELECT AVG(sal) AS avg\_sal

FROM A;

1. select id, avg(sal) as avg\_sal from A group by id;

SELECT dept\_name, COUNT(DISTINCT instructor.ID) AS instr\_count

FROM instructor, teaches

WHERE instructor.ID = teaches.ID

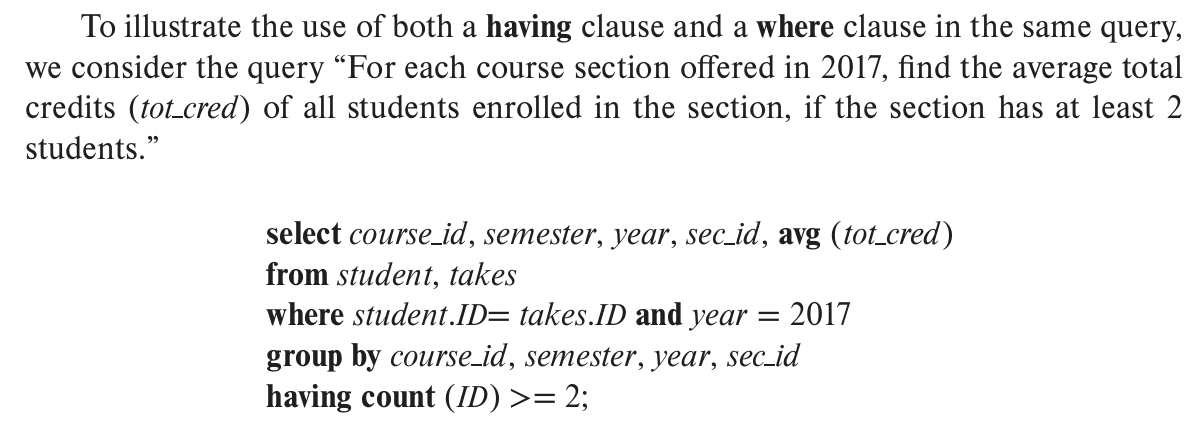
AND semester = 'Spring'

AND year = 2018

GROUP BY dept\_name;

Find the number of instructors in each department who teach a course in the Spring 2018 semester.

1. select dept\_name, ID, avg(salary) from instructor group by dept\_name,ID;
2. select dept\_name, avg(salary) as avg\_salary from instructor group by dept\_name having avg(salary) >42000;



1. select course\_id,semester,year,sec\_id,avg(salary) from instructor,teaches where teaches.ID=instrcutor.ID and year=2018 group by course\_id,semester,year having count(ID)>=2;
2. SELECT course\_id, semester, year, sec\_id, AVG(instructor.salary) AS avg\_salary

FROM instructor

JOIN teaches ON teaches.ID = instructor.ID

WHERE teaches.year = 2018

GROUP BY course\_id, semester, year, sec\_id

HAVING COUNT(teaches.ID) >= 2;

**PDF 3**

1. delete from instructor where id between 20 and 50;
2. delete from instructor where dept\_name='Finance';
3. delete from instructor where dept name in (select dept name from department where building = 'Watson');
4. delete from instructor where salary < (select avg (salary) from instructor);
5. Moregenerally, we might want to insert tuples on the basis of the result of a query. Suppose that we want to 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. We write:

insert into instructor select ID, name, dept name, 18000 from student where dept name = 'Music' and tot cred > 144;

1. insert into instructor select \* from instructor;
2. update instructor set salary= salary \* 1.05;
3. update instructor set salary = salary \*1.05 where salary < 70000;
4. As with insert and delete,anestedselect within an update statement may reference the relation that is being updated. As before, SQL first tests all tuples in the relation to see whether they should be updated, and it carries out the updates afterward. For example, we can write the request “Give a 5 percent salary raise to instructors whose salary is less than average” as follows:

update instructor set salary = salary \*1.05 where salary < (select avg (salary) from instructor);

1. update instructor set salary = case when salary <= 100000 then salary \*1.05 else salary \*1.03 end
2. UPDATE student

SET tot\_cred = (

SELECT SUM(credits)

FROM takes, course

WHERE student.ID = takes.ID

AND takes.course\_id = course.course\_id

AND takes.grade <> 'F'

AND takes.grade IS NOT NULL

);

1. select name, course id from student, takes where student.ID = takes.ID;

*This query can be written more concisely using the natural-join operation in SQL as:*

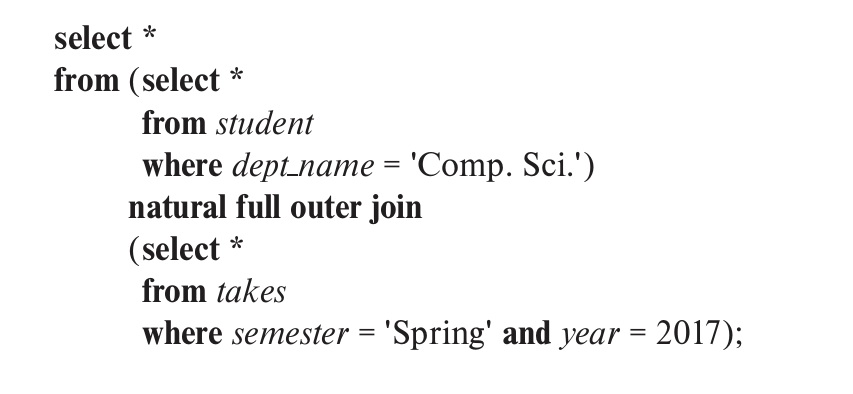
select name, course id from student natural join takes;

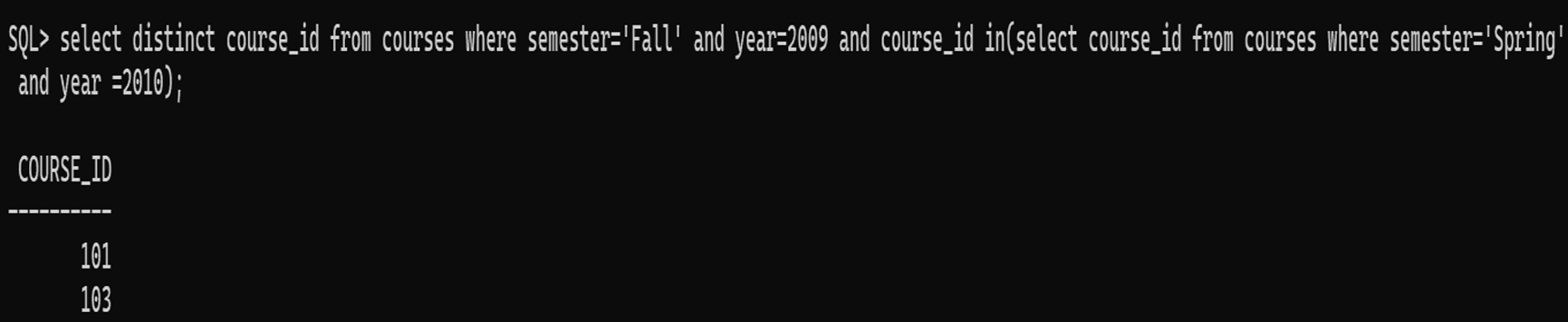
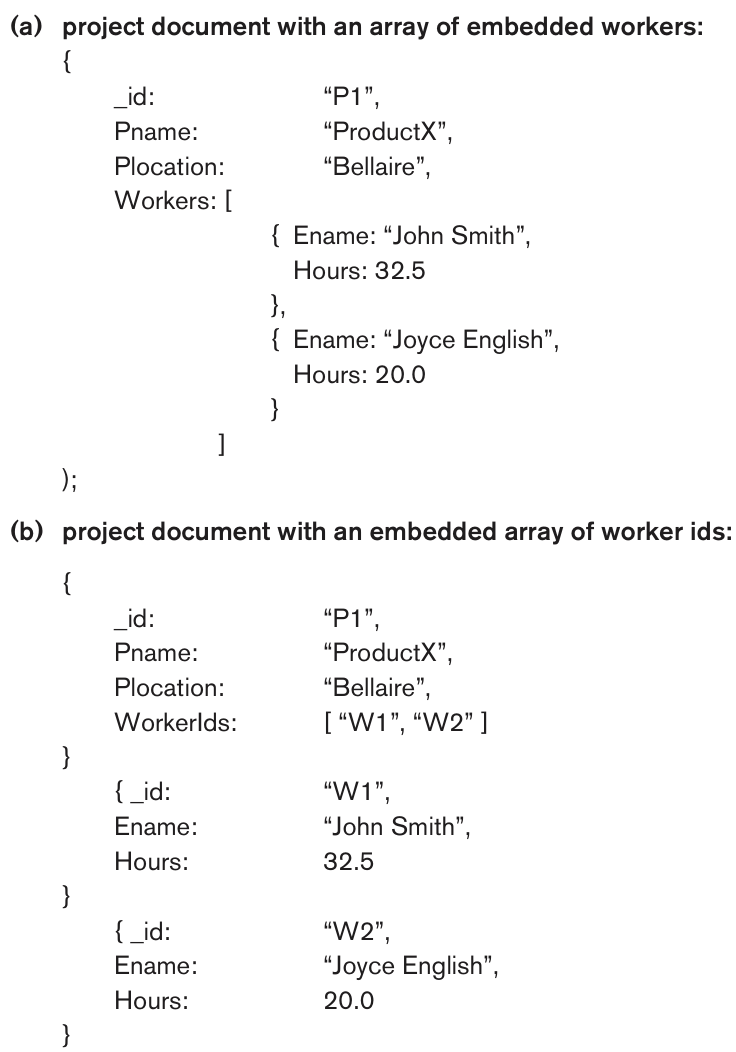
1. “List the names of students along with the titles of courses that they have taken.” The query can be written in SQL as follows:”

select name, title from student natural join takes, course where takes.course\_id = course.course\_id;

1. select name, title from (student natural join takes) join course using (course id);

course ID is a primary key

1. select \* from student natural join takes;
2. select \* from student natural left outer join takes;
3. select ID from student natural left outer join takes where course id is null;
4. select \* from takes natural right outer join student;
5. 
6. select \* from student left outer join takes on student.ID = takes.ID;
7. select \* from student left outer join takes on true where student.ID = takes.ID;
8. select \* from student join takes using (ID);
9. select \* from student inner join takes using (ID);
10. select distinct course id from section where semester = ’Fall’ and year= 2009 and course id in (select course id from section where semester = ’Spring’ and year= 2010);
11. from section where semester = ’Fall’ and year= 2009 and course id not in (select course id from section where semester = ’Spring’ and year= 2010);
12. select course id from section as S where semester = ’Fall’ and year= 2009 and exists (select \* from section as T where semester = ’Spring’ and year= 2010 and S.course id= T.course id);
13. select distinct S.ID, S.name from student as S where not exists ((select course id from course where dept name = ’Biology’) except (select T.course id from takes as T where S.ID = T.ID));
14. select T.course id from course as T where unique (select R.course id from section as R where T.course id= R.course id and R.year = 2009);
15. with max budget (value) as (select max(budget) from department) select budget from department, max budget where department.budget = max budget.value;



**What are CRUD Operations?**

CRUD operations represent the four essential operations for interacting with a database:

1. **Create**: Adds new data or documents to the database.
2. **Read**: Retrieves or queries existing data.
3. **Update**: Modifies or changes existing data.
4. **Delete**: Removes data or documents from the database.

In the context of NoSQL databases, CRUD operations allow developers to perform basic operations on collections and documents, often through APIs rather than traditional SQL queries.

**What is NoSQL?**

**NoSQL (Not Only SQL)** databases are designed to handle large volumes of structured, semi-structured, or unstructured data. They offer:

* **Flexibility**: Unlike relational databases, NoSQL uses dynamic schemas, making it suitable for diverse and evolving data.
* **Scalability**: Easily scales horizontally to handle massive datasets across distributed systems.
* **High Performance**: Optimized for rapid data operations and big data scenarios.
* **Varied Data Models**: Includes key-value stores, document databases, column-family stores, and graph databases.

**Types of NoSQL Databases**

1. **Key-Value Stores**: Simple storage using key-value pairs (e.g., Redis, DynamoDB).
2. **Document Stores**: Stores data as documents in formats like JSON or BSON (e.g., MongoDB, CouchDB).
3. **Column-Family Stores**: Optimized for large datasets (e.g., Cassandra, HBase).
4. **Graph Databases**: Specialized for interconnected data (e.g., Neo4j).

**Other Important Points from the PDF**

1. **Advantages of NoSQL**:
   * Handles massive volumes of data efficiently.
   * Adapts well to changes in the data model.
   * Suited for high-availability and distributed systems.
   * Avoids the constraints of relational tables and schemas.
2. **Comparison with SQL**:
   * **SQL Databases**: Use structured tables with predefined schemas; ideal for complex queries.
   * **NoSQL Databases**: Use schema-less or flexible structures; ideal for real-time and big data applications.
3. **Use Cases for NoSQL**:
   * Social media platforms.
   * Content management systems.
   * IoT data management.
   * Big data analytics.
4. **CAP Theorem**: NoSQL databases often prioritize two of the three: **Consistency**, **Availability**, or **Partition Tolerance**, depending on the use case.
5. **Examples of NoSQL Databases**:
   * MongoDB (document store).
   * CouchDB (document store with JSON storage).
   * Cassandra (highly scalable column-family store).
   * Neo4j (graph database for relationships).