**Database Systems**

**Lab 3**

**Key Constraints**

**The Tables**

There will be 3 tables in this session- Employee, Department and Dependent. The scenario is about an organization where records about their employees are kept by these 3 tables. The schema diagram is as follows.



Figure: Schema Diagram of the 3 Tables

**Primary Key and Foreign Key**

Every table in a database should have a primary key. Primary key is the column or group of columns in a table that uniquely identify every row of that table. Employee table has SSN as its primary key and Department table has DNUMBER as its primary key. If you take a look at the Dependent table, then you will find out two important aspects-

1. Primary key can be made of more than one columns
2. One table can have primary key that is actually a column of another table

The DNO of Employee table is actually the DNUMBER column of Department table. ESSN of Dependent table is the SSN column of Employee table. So, DNO is the foreign key of Employee table and ESSN is the foreign key of Dependent table.

You can declare primary key and foreign key during the creation of table of after the creation of table. Remember one thing- primary keys cannot be NULL.

1. **Keys during the Creation of Table**

CREATE TABLE table\_name(

column\_name1 datatype NOT NULL,

column\_name2 datatype,

……………………………….

column\_namen datatype,

PRIMARY KEY (column\_name1),

FOREIGN KEY (column\_name2) REFERENCES reference\_table\_name

);

Here, column\_name2 is actually a column of reference\_table\_name.

1. **Keys after the Creation of Table**

In this case, you can first build up the entire table and then can use ALTER TABLE command to declare primary key and/ or foreign key for that table.

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name

PRIMARY KEY (column\_name1);

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name

FOREIGN KEY (column\_name2) REFERENCES reference\_table\_name (column\_name\_in\_reference\_table);

**Demonstration**

Now, run the script provided this week to you named **L3A.sql**. The script creates the tables, key constraints and the data that the tables contain.

1. Try to delete Employee table by **“DROP TABLE employee;”** command. What does oracle say? As it contains a foreign key which is the primary key of Department table, it won’t allow you to drop Employee table.
2. Try to delete Department table as well by **“DROP TABLE department;”** command. Again, oracle won’t allow you to delete it.
3. **The only table you can delete among the 3 tables is Dependent table.**

From the schema diagram, you can understand easily why this is happening. **In a “Many to One” relation the table that is “Many side” only, can be dropped**. **But the table that is “one side” of a relation cannot be deleted.**

**So, if you want to delete all the tables, you need to drop dependent first, then employee and then department.**

Now, take a look at the first two tables- Department and Employee.

SELECT \* FROM department;

SELECT fname, lname, dno FROM employee;

You can see there are 4 entries in Employee table with department number 5. Try to execute following statements and observe what oracle is saying.

DELETE FROM department WHERE dnumber=5;

DELETE FROM employee WHERE dno=5;

Again take a look at the last two tables- Employee and Dependent.

SELECT ssn, fname, lname FROM employee;

SELECT essn, dependent\_name FROM dependent;

You can see that employee ssn 123456789 has 3 dependents from dependent table and from employee table, you can see that employee’s name is **John Smith.**

Now, try to execute the following statement and see what oracle is saying.

DELETE FROM employee WHERE ssn=’123456789’;

Now, run the script **L3B.sql.** Again try the following statements to be executed.

DELETE FROM department WHERE dnumber=5;

SELECT fname, lname, dno FROM employee;

SELECT \* FROM dependent;

What do you see? It was impossible with the script L3A**.sql.** We have deleted department number 5 from Department table, and then it deleted all the records from Employee and Dependent table that has department number 5 AUTOMATICALLY!

Only one thing is changed in our new script L3B**.sql**. ON DELETE CASCADE statement in the foreign key declaration. You have to take a look at that simple difference. It means- **if you delete data on the master table, all the related entries in detail table will be deleted automatically.**

You now have the facility to delete from the table on many side as well. Execute the following statement.

DELETE FROM employee WHERE dno=4;

Now, run the script **L3C.sql.** In this time, the statement in the foreign key is slightly changed- ON DELETE NO ACTION. It means, if you delete in the master table, related detail record will still be there. Execute the following statements.

DELETE FROM department WHERE dnumber=5;

SELECT fname, lname, dno FROM employee;

DELETE FROM employee WHERE ssn=’123456789’;

SELECT \* FROM dependent;

**UNIQUE Key**

UNIQUE keys are keys that are needed to be unique (not necessarily primary key) throughout the table. For example, if you have a table contains courses for an engineering department, then the name of the subjects do not need to be primary key but they need to be unique. The reason is- you will never ever should find two courses that have the same name. Run the script named **“L3D.sql”.** Take a look at the effect. It has UNIQUE identifier on course names. But one course name is attempted twice to insert. Take a look at the error message while running the script.

**CHECK and DEFAULT constraints**

Now, we will take a look at these two constraints. On the course table, say all of your courses’ pass marks are 40. Then why will you insert 40 in the passmark column every time? DEFAULT can save your time. Moreover, you may sometimes have to check if right data are inserted or not. What if anyone inserts a course with credit 5? (this is inappropriate for KUET!!). So, you need to check that. Run the script named “L3E.sql”. See how these two constraints are applied. Then try to violate these by **inserting inappropriate data (e.g., credit hours more than 4 or less than or equal to 0**. See what happens.

**Finally**

For other group, please drop all the tables-

DROP TABLE DEPENDENT;

DROP TABLE EMPLOYEE;

DROP TABLE DEPARTMENT;

**The SEECT Clause**

In this section, we will play with the most vital statement of SQL- SELECT statement. Remember, you can do so many things with this single statement along with others that a book can be written. There are many tricks that can be played with SELECT. So, try to catch more tricks.

The general syntax for SELECT statement is-

SELECT [DISTINCT | ALL]

{\* | [columnExpression [AS newName]] [,...] }

FROM TableName [alias] [, ...]

[WHERE condition]

[GROUP BY columnList] [HAVING condition]

[ORDER BY columnList]

In this case, only **SELECT and FROM are** manditory for this expression. All others are optional. Order of the clauses cannot be changed.

|  |
| --- |
| SELECT Specifies which columns are to appear in output. |
| WHERE Filters rows using condition or another query |
| GROUP BY Forms groups of rows with same column value. |
| HAVING Filters groups subject to some condition. |
| ORDER BY Specifies the order of the output |

**Demonstration**

Run the script L3B.sql and try to follow the steps provided below one by one and see the results. You can also take a note of them if you wish. Not all of the clauses used it SELECT statement will be covered in this section.

**All columns and all rows**

* SELECT fname, mi, lname, ssn, bdate, address, salary, superssn, dno

FROM employee;

* SELECT \* FROM employee;

Nb. In this case, “all columns” are substituted by “\*” sign.

**Specific columns and rows**

* SELECT fname, lname, dno

FROM employee;

Use of DISTINCT

DISTINCT is used to eliminate repeating elements.

* SELECT dno FROM employee;
* SELECT DISTINCT (dno) FROM employee;

You can see the difference between them.

**Calculated Fields**

You can make numerical calculations on related columns of a table also. In this case, we will divide the salary of employees who have department number 5.

* SELECT (salary/5) FROM employee WHERE dno=5;

Renamming column

When you are showing data by SELECT command, you can put the column name to be shown as your wish by **AS** clause. Remember, this naming is just for showing. It has no impact on the actual column name of the table.

* SELECT (salary/5) AS reduced\_salary FROM employee;

**Applying condition**

* SELECT fname FROM employee;
* SELECT fname FROM employee

WHERE dno>1;

See the difference.

SELECT fname, lname

FROM employee

WHERE dno=1 OR dno=5;

You can use AND operator also. See the result.

**Range search**

SELECT fname, lname

FROM employee

WHERE salary BETWEEN 40000 AND 50000;

Take a look at the BETWEEN clause here.

SELECT fname, lname

FROM employee

WHERE salary NOT BETWEEN 40000 AND 50000;

Take a look at the NOT BETWEEN clause here.

* SELECT fname, lname

FROM employee

WHERE salary>= 40000 AND salary <=50000;

Look at the range operators <= and >= here. These are more flexible than BETWEEN and NOT BETWEEN clauses.

**Set membership**

SELECT fname, lname

FROM employee

WHERE salary IN (30000, 40000);

This finds out the first and last name of those who has salary exactly 30000 or 40000. Similarly the following statement negates it.

SELECT fname, lname

FROM employee

WHERE salary NOT IN (30000, 40000);

Try to execute following statements one by one.

* SELECT fname, lname, address

FROM employee;

* SELECT fname, lname, address

FROM employee

WHERE address LIKE ‘%houston%’;

* SELECT fname, lname, address

FROM employee

WHERE address LIKE ‘%HOUSTON%’;

Found any difference? The second one does not work as the table has data in all capitals. That is why the third one works.

**Ordering by column values**

This time we will use ORDER BY clause. It is used to represent data in the tables in some particular/ desired order.

* SELECT fname, lname, salary, dno

FROM employee

ORDER BY salary;

* SELECT fname, lname, salary, dno

FROM employee

ORDER BY salary desc;

Which one is by default- ascending or descending?

When you are using multiple columns in ORDER BY clause, be careful. Because, sometimes it is difficult to see the difference in actual results.

* SELECT fname, lname, salary, dno

FROM employee

ORDER BY salary, dno;

In this case, the result will be-

|  |
| --- |
| FNAME LNAME SALARY DNO |
| ALICIA ZELAYA 25000 4 |
| AHMAD JABBAR 25000 4 |
| JOYCE ENGLISH 25000 5 |
| JOHN SMITH 30000 5 |
| RAMESH NARAYAN 38000 5 |
| FRANKLIN WONG 40000 5 |
| JENNIFER WALLACE 43000 4 |
| JAMES BORG 55000 1 |

SELECT fname, lname, salary, dno

FROM employee

ORDER BY salary, dno desc;

In this case the result will be-

|  |
| --- |
| FNAME LNAME SALARY DNO |
| JOYCE ENGLISH 25000 5 |
| ALICIA ZELAYA 25000 4 |
| AHMAD JABBAR 25000 4 |
| JOHN SMITH 30000 5 |
| RAMESH NARAYAN 38000 5 |
| FRANKLIN WONG 40000 5 |
| JENNIFER WALLACE 43000 4 |
| JAMES BORG 55000 1 |

You can find out the difference in the first 3 rows where salary is all 25000. take a look at the change in ordering regarding to the department number.

**Appendix: PRIMARY KEY vs UNIQUE KEY**:

|  |  |  |
| --- | --- | --- |
|  | **PRIMARY KEY** | **UNIQUE KEY** |
| NULL | It doesn’t allow Null values. Because of this we refer PRIMARY KEY = UNIQUE KEY + Not Null CONSTRAINT | Allows Null value. But only one Null value. |
| INDEX | By default it adds a clustered index | By default it adds a UNIQUE non-clustered index |
| LIMIT | A table can have only one PRIMARY KEY Column[s] | A table can have more than one UNIQUE Key Column[s] |
| CREATE SYNTAX | Below is the sample example for defining a single column as a PRIMARY KEY column while creating a table:  CREATE TABLE dbo.Customer ( Id INT NOT NULL PRIMARY KEY, FirstName VARCHAR(100), LastName VARCHAR(100), City VARCHAR(50) )  Below is the Sample example for defining multiple columns as PRIMARY KEY. It also shows how we can give name for the PRIMARY KEY:  CREATE TABLE dbo.Customer ( Id INT NOT NULL, FirstName VARCHAR(100) NOT NULL, LastName VARCHAR(100), City VARCHAR(50), CONSTRAINT PK\_CUSTOMER PRIMARY KEY (Id,FirstName) ) | Below is the sample example for defining a single column as a UNIQUE KEY column while creating a table:  CREATE TABLE dbo.Customer ( Id INT NOT NULL UNIQUE, FirstName VARCHAR(100), LastName VARCHAR(100), City VARCHAR(50) )  Below is the Sample example for defining multiple columns as UNIQUE KEY. It also shows how we can give name for the UNIQUE KEY:  CREATE TABLE dbo.Customer ( Id INT NOT NULL, FirstName VARCHAR(100) NOT NULL, LastName VARCHAR(100), City VARCHAR(50), CONSTRAINT UK\_CUSTOMER UNIQUE (Id,FirstName) ) |
| ALTER SYNTAX | Below is the Syntax for adding PRIMARY KEY CONSTRAINT on a column when the table is already created and doesn’t have any primary key:  ALTER TABLE dbo.Customer ADD CONSTRAINT PK\_CUSTOMER PRIMARY KEY (Id) | Below is the Syntax for adding UNIQUE KEY CONSTRAINT on a column when the table is already created:  ALTER TABLE dbo.Customer ADD CONSTRAINT UK\_CUSTOMER UNIQUE (Id) |
| DROP SYNTAX | Below is the Syntax for dropping a PRIMARY KEY:  ALTER TABLE dbo.Customer DROP CONSTRAINT PK\_CUSTOMER | Below is the Syntax for dropping a UNIQUE KEY:  ALTER TABLE dbo.Customer DROP CONSTRAINT UK\_CUSTOMER |