

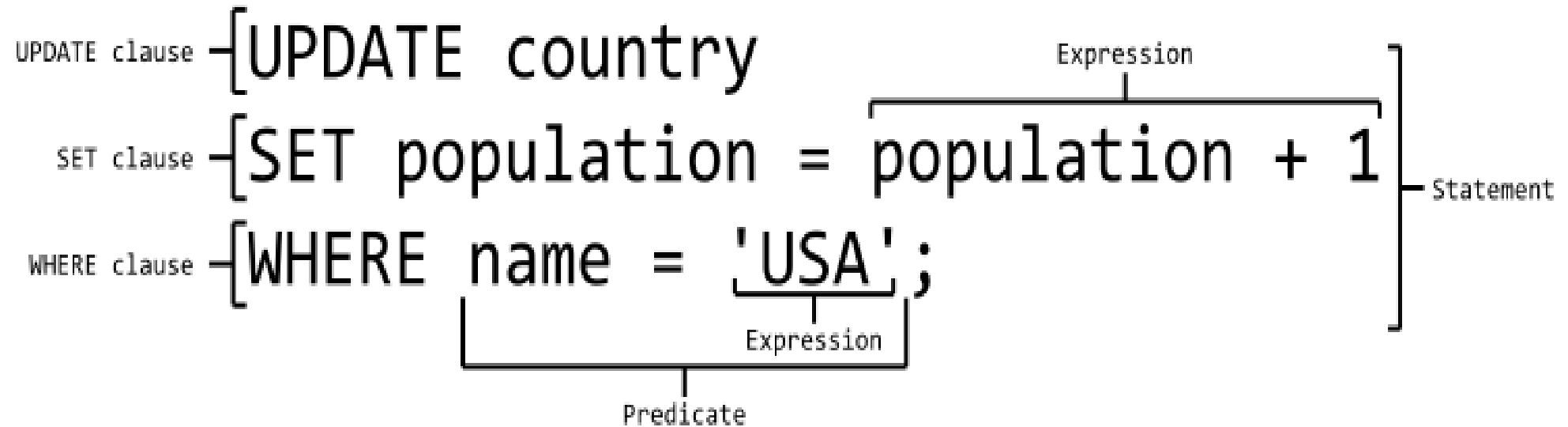
Unit 3

Structured Query Language

Overview of SQL Query Language

- IBM developed the original version of SQL, originally SEQUEL in 1970s
- The sequel language has evolved since then and the name changed as SQL and has established itself as the standard relational database language
- In 1986, the ANSI and ISO published an SQL standard called SQL-86
- Recently SQL:2008
- **SQL (Structured Query Language)** is a special – purpose programming language designed for managing data held in a relational database management system(RDBMS), or for stream processing in a relational data stream management system(RDSMS).

SQL Language Elements



- SQL statements also include the semicolon (";") statement terminator.

Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- ❑ The **schema** for each relation/table.
- ❑ The **domain of values** associated with each attribute.
- ❑ **Integrity constraints**
- ❑ And as we will see later, also other information such as
 - ❑ 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.

Domain Types in SQL

- ❑ **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).
- ❑ **numeric(*p*,*d*)**. Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point.
- ❑ **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- ❑ **float(*n*)**. Floating point number, with user-specified precision of at least *n* digits.

Built-in Data Types in SQL

□ **date**: Dates, containing a (4 digit) year, month and date

□ Example: **date** '2005-7-27'

□ **time**: Time of day, in hours, minutes and seconds.

□ Example: **time** '09:00:30' **time** '09:00:30.75'

□ **timestamp**: date plus time of day

□ Example: **timestamp** '2005-7-27 09:00:30.75'

□ **interval**: period of time.

In Oracle this data type is used as below-

Example: CREATE TABLE Emp (empno NUMBER, ename VARCHAR2(50), job VARCHAR2(255) ,
year_of_experience INTERVAL YEAR TO MONTH);

INSERT INTO EMP VALUES (1,'Rajesh','S.Manager', INTERVAL '10-5' YEAR TO MONTH);

Oracle- SQL Data Types...

1. Character

- **Char** – fixed length character string that can varies between 1-2000 bytes
- **Varchar / Varchar2** – variable length character string, size ranges from 1-4000 bytes.
- **Long** - variable length character string, maximum size is 2 GB

Example: **Name** **Char(10)**

2. **Number** : Can store +ve,-ve,zero,fixed point,floating point with 38 precision.

- **Number** – {p=38,s=0}
- **Number(p)** - fixed point
- **Number(p,s)** –floating point (p=1 to 38,s= 84 to 127)

Example: **Marks** **Number(3)** **fixed point**
 Salary **Number(9,2)** **Floating point**

SQL Data Types

3. **Date** : used to store date and time in the table. DB uses its own format of storing in fixed length of 7 bytes for century, date, month, year, hour, minutes, seconds. The default data type is “dd-mon-yy” **Example:** Birth_date **Date**

4. **Interval Year To Month** : Stores a period of time using the YEAR and MONTH date time fields **Example:** year_of_experience **INTERVAL YEAR TO MONTH**

5. **Raw Datatype**: used to store byte oriented data like binary data and byte string. Mainly used when moving data between different systems. Oracle Recommends to store as BLOB **Example:** blob_data **BLOB**

6. **Other** :

- CLOB – A character large object containing single-byte or multi byte characters.
- BLOB – stores large binary objects such as graphics, video, sounds..
- BFILE – Contains a locator to a large binary file stored outside the database.

Different Types of Commands

- ✓ **DDL commands: -**

To create and modify database objects - CREATE, ALTER, DROP

- ✓ **DML commands: -**

To manipulate data of a database objects- INSERT, DELETE, UPDATE

- ✓ **DQL command: -**

To retrieve the data from a database - SELECT

- ✓ **DCL commands: -**

To control the data of a database – GRANT, REVOKE

- ✓ **TCL commands:-**

To control and manage transactions – COMMIT, SAVEPOINT,

Create Table Construct

- An SQL relation is defined using the **create table** command:

create table r ($A_1 D_1, A_2 D_2, \dots, A_n D_n$); both are equivalent syntax

CREATE TABLE *table-name* (*column_name* Datatype(size),
column_name Datatype (size), . . .);

- r is the name of the relation/table
- each A_i is an attribute (column) name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i
- **Example:**

```
create table instructor (  
    ID          char(5),  
    name        varchar(20),  
    dept_name varchar(20),  
    salary     numeric(8,2));
```

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **insert into** *instructor* **values** ('10211', **null**, 'Biology', 66000);

INTEGRITY CONSTRAINTS

- ❑ Valid data means –the data which follows certain **rules/ regulations of real world system**.
- ❑ Therefore designer has to ensure that data entered by user has to be checked against these rules and allowed to **store if valid otherwise** need to be **rejected**.
- ❑ **Integrity constraints guard against accidental damage** to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- ❑ **Example:**
 - ❑ Data in some Column such as *Phone_Number* is **mandatory** for user to enter.
 - ❑ Data in some Column such as *Registration_Number* has to be **Unique** (No duplicated allowed).
 - ❑ Data in some Column such as *Registration_Number* is used **to identify every student** distinctly.
 - ❑ **Valid range of Data** for some Column such as *Under_Gradate* is BSc, B.Tech. BE.
 - ❑ A SB account must have a **balance greater than 1000/-**

TYPE of CONSTRAINTS

- **Rule/Constraints** can be imposed on **single column** or **combination of columns**.
 - **Column-level Constraints**- Imposed on **Single Column**. Defined along with **Column**
 - **Table Level Constraint**.- Defined at the end after defining all the columns.
 - ▶ **Multi-level Column**.
 - **Primary key** imposed on combination of columns- (**Name,F.name,Surname**)
 - ▶ **Constraint imposed on a column that reference another column** in the constraint.
 - Assume that are two columns in the table say- **Date_of_Birth** and **Date_of_Join**.
 - We want to impose condition(constraint) on **Date_of_Join** that

Date_of_Join > Date_of_Birth.

Integrity Constraints in Create Table

SQL supports a number of different integrity constraints.

- **not null -**
- **primary key** (A_1, \dots, A_n)
- **foreign key** (A_m, \dots, A_n) **references** r
- **Unique**
- **Check**
- **Default**

NOT NULL

- **NULL** is special kind of value applicable to any domain(datatype).
 - Note: NULL is **not equivalent** to "" or ' '
- In some cases, value to some column is **mandatory to enter**.
- In other words we want to **force the user to enter** some value to the column.

Example: Assume that for the table Instructor we want to make user to enter some values for name

```
create table instructor (  
    ID          char(5),  
    name        varchar(20) NOT NULL , dept_name varchar(20),  
    salary      numeric(8,2) );
```

PRIMARY KEY...

- Identifies every tuple(record/row) in the table uniquely.
- **primary key** ($A_{j1}, A_{j2}, \dots, A_{jm}$)
 - Where $A_{j1}, A_{j2}, \dots, A_{jm}$ are the set of attributes in the table used to form a primary key.
 - $A_{j1}, A_{j2}, \dots, A_{jm}$ are said to be components of primary key.
 - Primary key may be imposed on a **single attribute** **or** **multiple attributes** of the table.
- There can be **ONLY ONE PRIMARY** key for a table.
- **Properties:**
 - **NO component** of primary key can be **NULL**.
 - Values to the columns must be **Unique**(Duplicate values can't be entered to a column)

Example: Declare *ID* as the primary key for *instructor*

```
create table instructor (  
    ID          char(5) PRIMARY KEY  
    name        varchar(20) not null,  
    dept_name   varchar(20),  
    salary      numeric(8,2));
```

...PRIMARY KEY-Table Level

- ❑ **Example:** Create a table Enrollment containing fields –**SID** –student ID , **CNo**-Course Number **and** **Year** – Joining Year to the Course.
- ❑ Condition to be imposed that – We want to identify a student Uniquely who enrolled to a Course on a Particular year. Therefore combination of SID,CNO and YEAR has to be Unique and can't be Null.
- ❑ Therefore we need to impose Primary Key on SID,CNO and YEAR .
- ❑ Since Constraint is on multiple column, it has to be defined as Table level Constraints.

CREATE TABLE Enrollment

(SID char(9) NOT NULL,

CNO varchar2(7) NOT NULL,

Year number(2) NOT NULL,

Grade char(2),

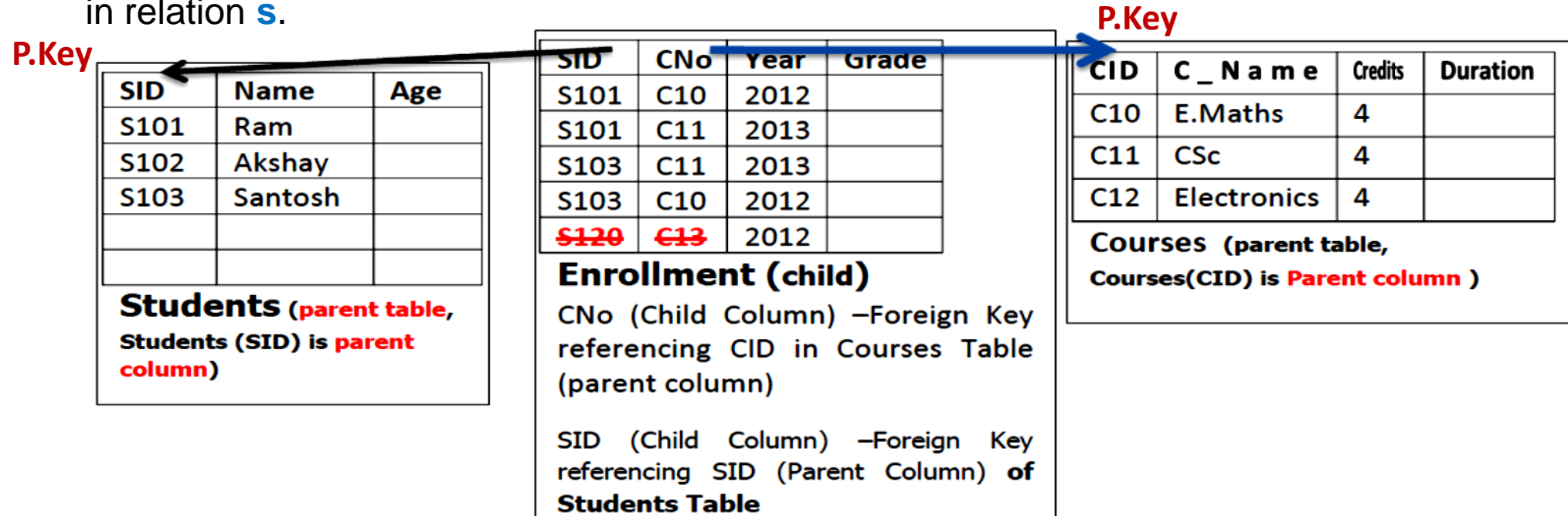
PRIMARY KEY (SID, CNO, Year));

Note: primary key defined after defining all the columns

Note: **NO component** of primary key can be **NULL**

FOREIGN KEY...

- foreign key ($A_{k1}, A_{k2}, \dots, A_{kn}$) references s :
- The foreign key in a relation r specification says that the values of attributes ($A_{k1}, A_{k2}, \dots, A_{kn}$) for any tuple in the relation r must correspond to values of the primary key attributes of some tuple in relation s .



Enrollment can be done only to those who are student, therefore **SID** column in **Enrollment** can have only values which are present in **SID** in **Student** table.

This condition is imposed by defining **SID** in **Enrollment** as Foreign key referencing **Students**

This is known as Referential Integrity Constraint

...Referential Integrity Constraint

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
 - **Example:** If “S101” is a **Student Id** appearing in one of the tuples in the **Enrollment** relation, then there exists a tuple in the **Students** relation for “S101”.
- Let **A** be a **set of attributes**. Let **R** and **S** be two relations that contain attributes **A** and where **A** is the **primary key of S**. **A** is said to be a **foreign key of R** if for any **values of A appearing in R** these values **also appear in S**.

Child	R A- is Foreign key				S - A is Primary Key				Parent
	Q	P	A	A	B	C	
			a2		a1				
			a3		a2				
			a5		a3				
			a2		a4				
			a3		a5				

Note: In relation **R**, attribute **A** can't contain a value which is not existing in attribute **A** of relation **S**. In the example above , at this instance **A** in **R** can't have a value **a6** or **a7** etc.

..FOREIGN KEY

□ Properties:

- A **Foreign key** can contain-
 - ▶ Only values present in the corresponding Parent Column/s.
 - ▶ **NULL** values accepted, if Foreign key is not defined with additional NOT NULL constraints.
- Foreign key column can reference to any column (parent column) **whose data type, width is same** and **Parent column** has to be defined with **Primary key** or **Unique constraint**.
- A **Parent Column has to exist before creation of Child Column** with Foreign key Constraint.

Restrictions: Any **UPDATE/INSERT/DELETE** of Records , **ALTER** or **DROP**

Operation that Violates any of the above properties is restricted and hence

Rejected by the Database System.

..FOREIGN KEY column-level

□ Example:

□ We have to create both **Parent Tables** First.

□ CREATE TABLE Students (SID char (9) **PRIMARY KEY** , Name varchar2(25) not null, Age integer);

□ CREATE TABLE Courses (CID varchar2 (9) **UNIQUE** , C_Name varchar2(25) not null, Credits number(2), Duration Number(2));

□ After Creating Parent Table/s, create Child tables

□ CREATE TABLE Enrollment

(SID char (9) **NOT NULL References Students**,

CNo varchar2 (9) **References Courses(CID)**,

Year number (2) **not null**,

Grade char (2), Primary key (SID, CNO, Year));

Why- Courses(CID)?

..FOREIGN KEY table-level

□ Example:

ITEMS			TRANSACTIONS			
Primary Key			Foreign Key			
ITEM_NAME	COMP_NAME	PRICE	IT_NAME	COMP_NAME	TR_DATE	QTY
Brush	Colgate	50	Brush	Oral-B	27-07-2019	10
Brush	Oral-B	60	Paste	DaburRed	28-07-2020	5
Paste	Colgate	90	Brush	Colgate	28-07-2021	18
Paste	DaburRed	87	Brush	Oral-B	29-07-2019	16

□ Parent(Master) Table:

```
CREATE TABLE Items( Item_name varchar2(10), Comp_name varchar2(10),  
Price Number(3),  
PRIMARY KEY ( Item_name,Comp_name ) );
```

□ Child(Detail) Table

```
CREATE TABLE Transactions( It_name varchar2(10), Comp_name varchar2(10),  
Tr_Date date, Qty Number(3),  
FOREIGN KEY(It_name, Comp_name) REFERENCES Items);
```

Does the following table get created with Foreign key constraint?

- Create table DEPT (Dno Varchar2(3) ,Dname varchar2(10));
- Create table EMP(Empno Number(3) Primary key, Name varchar2(10), Deptno varchar2(3) References Dept);

ERROR: referenced table does not have a primary key

Does the following table get created with Foreign key constraint?

- Create table DEPT (Dno Varchar2(3) UNIQUE ,Dname varchar2(10));
- Create table EMP(Empno Number(3) Primary key, Name varchar2(10), Deptno varchar2(3) References Dept);

No: referenced table has a unique key so it has to be refered during foreign key definition

Write the SQL commands to create following tables with mentioned constraints

*Assume that one student stays in one particular room of one particular Hostel only.

Student

Column	DataType	Constraint
RegNo	Number	Primary key
Name	Varchar	
Phone	Number	Unique

Hostel

Column	DataType	Constraint
Hostel_No	Varchar	Primary Key
Room_No	Number	Primary Key
RegNum		Foreign Key

Create table Student(RegNo Number(3) PRIMARY KEY, Name Varchar2(10), Phone Number(10) UNIQUE);

Create table Hostel (Hostel_no Varchar2(5), Room_no Number(3), Reg_no Number(3) REFERENCES Student, **PRIMARY KEY(Hostel_no, Room_no)**);

Restrictions on **INSERT / UPDATE / DELETE** Operations Over Foreign Key

Any **INSERT / UPDATE / DELETE** of Records , **ALTER** or **DROP** Operation that Violates any of the Foreign key properties is restricted and hence the operation is Rejected by the Database System.

..INSERT

Syntax-

INSERT INTO table_name(column1,column2,..) VALUES (value1,value2,....)

Example: Insert a record into Course table having values to Course_id, Dept_Name columns only. Course(Course_id,title,Dept_Name,Credits)

insert into course values ('CS-438', NULL, 'Comp. Sci.', NULL);

Note: NULL is not same as 'NULL'

UPDATE

To modify any column/s value in a already existing record.

Syntax:

```
UPDATE table_name SET column1=value1,column2=value2,...  
WHERE condition involving any of column/s in the table ;
```

Example: Consider the table Instructor(Id, Name, Dept_name, Salary).
Increase the salary of instructor with ID I201 by 10%.

```
UPDATE Instructor SET Salary=Salary+Salary*0.1 WHERE Id='I201';
```

DELETE

Syntax:

DELETE FROM table_name WHERE condition;

Example:

- Delete all instructors

delete from *instructor*

- Delete all instructors from the Finance department

delete from *instructor*

where *dept_name*= 'Finance';

..FOREIGN KEY – INSERT Restrictions

EMP- DEPTNO Foreign Key			DEPARTMENT - DNO Primary Key		
EMPNO	NAME	DEPTNO	DNO	NAME	BUDGET
100	Raj	D1	D1	MCA	128999
101	Krishna	D2	D2	CompSc	124456
102	Manoj	D1	D3	Mech	123562
103	Ravi	D3			
104	Shriivas				

❑ INSERT INTO EMP VALUES(105,'Rajesh','D4');

Is rejected, to execute above INSERT command, execute in following Order

❑ INSERT INTO DEPT VALUES('D4','Physics',125678);

Note-Parent record is added to DEPARTMENT and now we can add Employee with D4 department

❑ INSERT INTO EMP VALUES(105,'Rajesh','D4'); **Now it is Accepted.**

..FOREIGN KEY- UPDATE/DELETE Restrictions

EMP- DEPTNO Foreign Key			DEPARTMENT - DNO Primary Key		
EMPNO	NAME	DEPTNO	DNO	NAME	BUDGET
100	Raj	D1	D1	MCA	128999
101	Krishna	D2	D2	CompSc	124456
102	Manoj	D1	D3	Mech	123562
103	Ravi	D3			
104	Shriivas				

- Similarly,
- UPDATE EMP SET DEPTNO='D5' WHERE EMPNO=100;
is **Rejected**.
- UPDATE EMP SET DEPTNO='D3' WHERE EMPNO=100;
is **Accepted**.
- DELETE FROM DEPARTMENT WHERE DNO= 'D1';
is **Rejected**

To execute above DELETE command, execute in following Order

- 1st Delete from **Child** Table(EMP) and then 2nd Delete from **Parent**(DEPARTMENT)
 - This Deletion process can be **automated** by using Clause **ON DELETE CASCADE / ON DELETE SET NULL** while creating Child Table
- Similarly Altering Structure of DNO or Dropping DNO is **Rejected**.

..FOREIGN KEY- ON DELETE CASCADE/ON DELETE SET NULL

- A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in Oracle.

- **Example:** Create tables give in [slide 29](#) with **ON DELETE CASCADE** clause along with FOREIGN KEY.

- **Parent(Master) Table:**

- **CREATE TABLE Department (Dno varchar(2) PRIMARY KEY, Name varchar(10), Budget Number(9));**

- **Child(Detail) Table**

- **CREATE TABLE Emp (Empno number(3) PRIMARY KEY, Name varchar(10), Deptno varchar(2) REFERENCES Department ON DELETE CASCADE);**

Any Delete operation on the table Department(Parent) first deletes dependent records in the EMP(child) table automatically. Thus Delete operation restriction on Foreign key constraint is get resolved automatically.

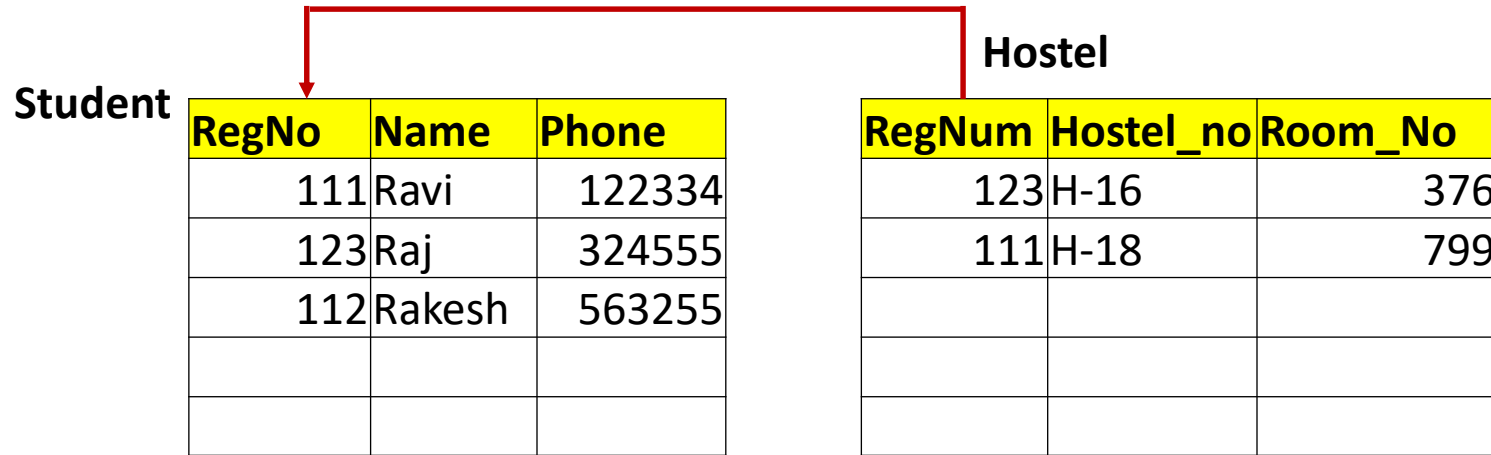
..FOREIGN KEY- ON DELETE CASCADE/ON DELETE SET NULL

- A foreign key with “**ON DELETE SET NULL**” means that if a record in the parent table is deleted, then the corresponding records in the child table will have the **foreign key fields set to null**. The records in the child table will **not be deleted**.
- **Example:** Create tables give in [slide 18](#) with **ON DELETE SET NULL** clause along with FOREIGN KEY.
- **Parent(Master) Table:**
 - **CREATE TABLE Department**(Dno varchar(2) PRIMARY KEY, Name varchar(10), Budget Number(9));
- **Child(Detail) Table**
 - **CREATE TABLE Emp**(Empno number(3) PRIMARY KEY, Name varchar(10), Deptno varchar(2) **REFERENCES Department ON DELETE SET NULL**);

..FOREIGN KEY- ON DELETE CASCADE/ON DELETE SET NULL

- When a record is deleted from Department(Parent) table it will not delete dependent records in the EMP(child) table instead puts NULL values to corresponding foreign key column/s.
- Thus removes dependency of corresponding records in the child table on table records being deleted in the Parent table.
- Thus Delete operation restriction on Foreign key is get resolved automatically.

Exercise



Note: RegNo-P.key ;

(Hostel_No,Room_No)- P.Key

RegNum- F.Key

What is the result of execution of following SQL statements?

INSERT INTO Student VALUES(115,'Ajay',567899);

INSERTED

INSERT INTO Student VALUES(112,'Sridhar',89979);

NOT-INSERTED

INSERT INTO Hostel VALUES(112,'H-16',376);

NOT INSERTED

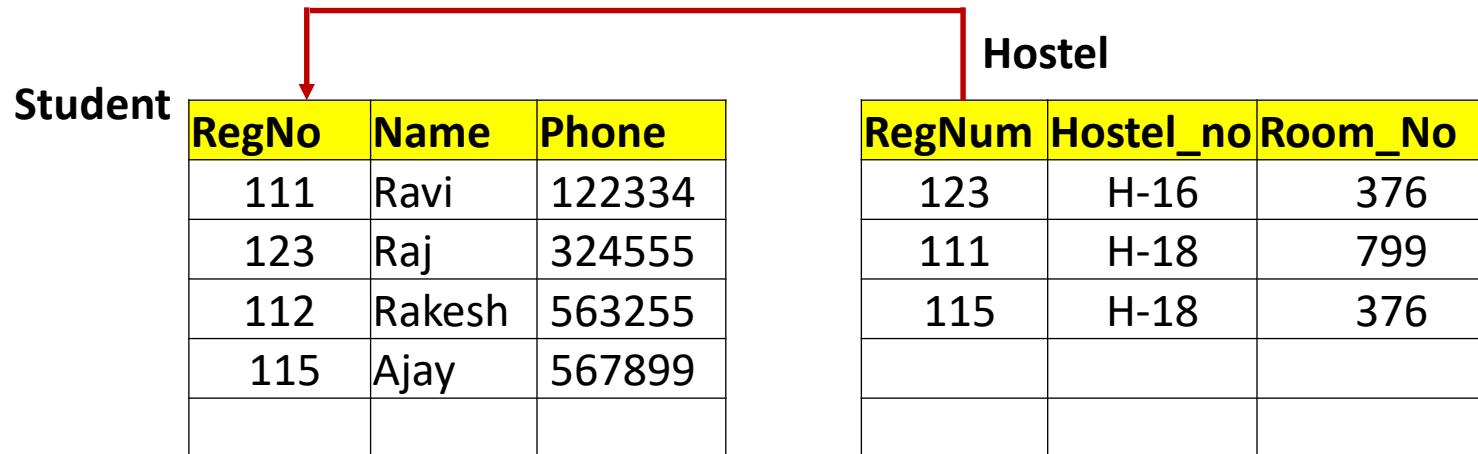
INSERT INTO Hostel VALUES(113,'H-17',234);

NOT-INSERTED

INSERT INTO Hostel VALUES(115,'H-18',376);

INSERTED

Exercise



Note: RegNo-P.key ;

(Hostel_No,Room_No)- P.Key

RegNum- F.Key

What is the result of execution of following SQL statements?

UPDATE Student SET Regno=113 WHERE Regno=112;

UPDATED

UPDATE Student SET Regno=222 WHERE Regno=123;

NOT-UPDATED

UPDATE Hostel SET RegNum=113 WHERE RegNum=123;

UPDATED

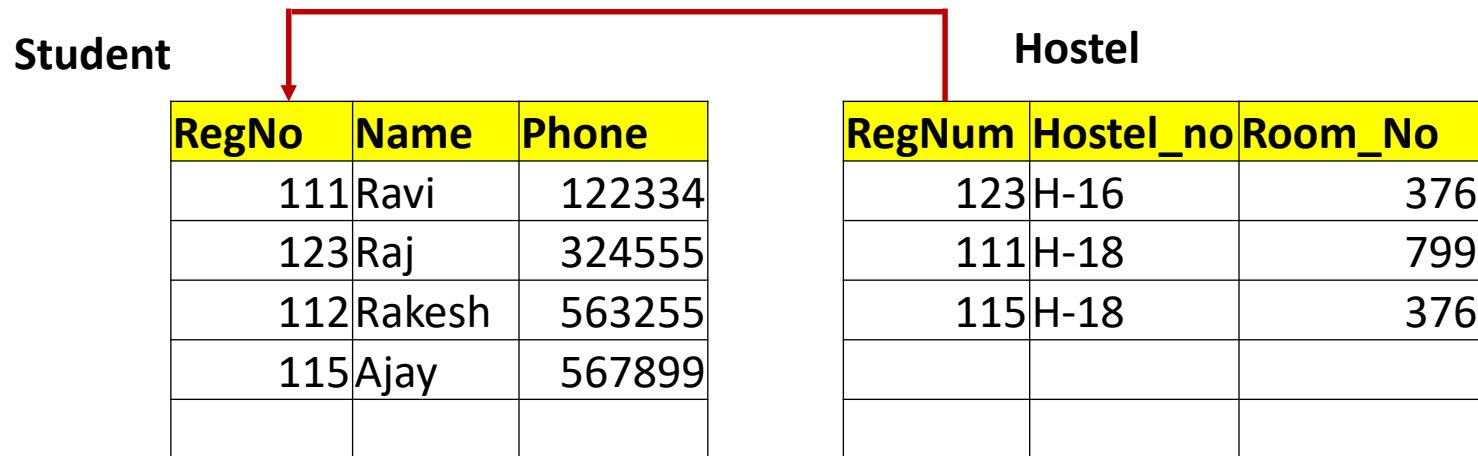
UPDATE Hostel SET RegNum=null WHERE RegNum=111;

UPDATED

UPDATE Hostel SET RegNum=118 WHERE RegNum=111;

NOT-UPDATED

Exercise



Note: RegNo-P.key ;

(Hostel_No,Room_No)- P.Key

RegNum- F.Key

What is the result of execution of following SQL statements?

DELETE FROM Student WHERE Regno=112;

DELETED

DELETE FROM Student WHERE Regno=123;

NOT-DELETED

DELETE FROM Hostel WHERE Regnum=123;

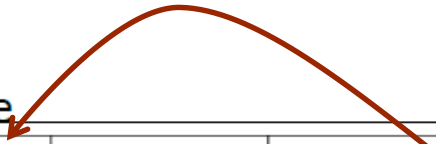
DELETED

DELETE FROM Student WHERE Regno=123;

DELETED

..FOREIGN KEY - Recursive Relationship

EMP table



EMPNO	ENAME	MGRNO
100		103
101		100
103		104
104		104
105		

MGRNO is the Employee number of Manger. Employee with EMPno 103 is the Manger for Employee with Empno 100. Therefore MGRNO is Foreign Key Referencing EMPNO

Example:

```
CREATE TABLE EMP ( Empno number(3) PRIMARY KEY, Ename  
Varchar2(10), MGRNO number(3) );
```

Note: Referential Integrity constraint on MGRNO can be defined using
Alter Table command **after creating EMP table**

OR

```
CREATE TABLE EMP( Empno number(3) PRIMARY KEY, Ename  
Varchar2(10), MGRNO number(3) REFERENCES EMP );
```

Exercise

Create a table **Student** (Regno, Name, Class_Representative)

Where RegNo is Primary key and Class_Representative is RegNo of students who are Class Representatives. Assume proper data type and size.

```
CREATE TABLE Student(Regno Number(3) PRIMARY KEY, Name Varchar2(10),  
Class_Representative Number(3) References Student);
```

Inserting Data into Student table having Recursive relationship

```
CREATE TABLE Student (Regno Number(3) PRIMARY KEY, Name  
Varchar2(10), Class_Representative Number(3) References Student);
```

- INSERT INTO Student VALUES(123,'AAAA',122);

Error

ORA-02291: integrity constraint (MCA2020.SYS_C007551) violated - parent key not

- INSERT INTO Student VALUES (123,'AAAA',NULL);
- INSERT INTO Student VALUES(122,'AAAA',NULL);
- UPDATE Student SET Class_Representative=122 WHERE Regno= 123;

OR

INSERT INTO Student VALUES(122,'AAAA',NULL); followed by INSERT INTO Student VALUES (123,'AAAA',122);

UNIQUE...

□ **unique** (A_1, A_2, \dots, A_m)

- The unique specification states that the attributes A_1, A_2, \dots, A_m form a candidate key.
- Candidate keys are **permitted to be null** (in contrast to primary keys).

□ **Example:**

```
CREATE TABLE Student(  
    ID varchar(5) PRIMARY KEY,  
    Name Varchar(10),  
    Phone number(10) UNIQUE,  
    tot_credit Number(2) );
```

Phone is implemented with **Column level UNIQUE** Constraints.

Exercise

UNIQUE...

Answer the validity of following statements with respect UNIQUE constraint on ID column-

INSERT INTO Student VALUES(123,'Vinay',7799889788,54) ; YES/~~NO~~

INSERT INTO Student VALUES(123,'Vinay',7799889788,54); ~~YES~~/NO

INSERT INTO Student VALUES (NULL,'Vinay',7799999788,54); ~~YES~~/NO

INSERT INTO Student VALUES (124,'Raj',NULL,54); YES/~~NO~~

..UNIQUE

- In the following table combination of **Area_code & Phone_Num** is **Unique** for a landline phone.
- **Area_code & Phone_Num** is to be implemented as **Table-level Constraint**,
- **Example:**

```
CREATE TABLE BsnL_Customer (  
    Customer_ID number(7) PRIMARY KEY,  
    Name varchar(10) NOT NULL,  
    Address varchar(20),  
    Area_Code Number(4),  
    Phone_Num Number(6),  
    UNIQUE(Area_Code , Phone_Num ) );
```

Exercise

Create a table **Customer**(Cust_id, Name, Phone, Email, Policy_No) **Cust_id**, **Phone, Email & Policy_No** contains **unique values** and assume Cust_id as Primary key. Also make **Phone number mandatory**.

Assume proper data type and size

```
CREATE TABLE Customer(Cust_id Varchar2(5) PRIMARY KEY ,  
Name Varchar2(10), Phone Varchar2(10) UNIQUE NOT NULL,  
Email Varchar2(20) UNIQUE , Policy_No Varchar2(10) UNIQUE );
```

The CHECK clause – Using IN

□ check (P)

where **P** is a predicate(condition)

Example: Ensure that Type of Courses offered by a department is any one of MCA, MTech, BTech, MS.

```
CREATE TABLE Department (  
    Department_name varchar2 (8) PRIMARY KEY,  
    Course_Type varchar2 (8) CHECK( Course_Type IN( 'MCA',' MTech ',' BTech', 'MS')),  
    Numb_of_Semester Number(1),  
    In_take_stud_num Number(2),  
    Department_Phone Number(10) NOT NULL UNIQUE );
```

Note: **IN** works like a **Belongs to a set** Operator

User_enetred_value **∈** { 'MCA',' MTech',' BTech', 'MS' } , evaluates to **TRUE** or **FALSE**

..The CHECK clause –Using BETWEEN

- ❑ Create table *Instructor* and ensure that **Salary** column accepts only values in the range **50000 to 200000** (both upper and lower bound values are valid).

❑ CREATE TABLE *instructor* (

<i>ID</i>	char(5),
-----------	-----------------

```
name          varchar2(20),
```

dept_name **varchar2(20),**

salary **number(8,2)** **CHECK(Salary>=50000 AND Salary<=200000)**

);

❑ CREATE TABLE *instructor* (

ID **char(5),**

name **varchar2(20),**

dept_name **varchar2**(20),

```
salary number(8,2) CHECK( Salary BETWEEN 50000 AND 200000) );
```

..The check clause - using LIKE (Pattern Matching)

Example:

- ❑ Create a table CANDIDATES(**CandtId**, Name, Branch) appearing for entrance exam at MIT.
Candidate numbers must be Unique & every **candidate number** must **start with MIT**.
- ❑ CREATE TABLE CANDIDATES(CandtId varchar2(7) PRIMARY KEY **CHECK (CandtId LIKE 'MIT%')**, Name varchar2(10), Branch varchar2(10));
- ❑ INSERT INTO CANDIDATE VALUES('MIT1020', 'Raghu', 'CompSc'); Accepted
- ❑ INSERT INTO CANDIDATE VALUES('MIIT1021', 'Ram', 'CompSc'); Rejected

Wild characters-

% any number of characters

_ (underscore) Single character

..The check clause - using function UPPER()

Example:

- ❑ Create a table CANDIDATES(CandtID, Name, **Branch**) appearing for entrance exam at MIT.
Candidate numbers must be Unique & every candidate number must start with MIT. User must enter **Branch in Capital letters only**.
- ❑ CREATE TABLE CANDIDATE(CandtId varchar2(7) PRIMARY KEY CHECK (CandtId LIKE 'MIT%'),
Name varchar(10),Branch varchar(10) **CHECK(Branch=UPPER(Branch))**);
- ❑ **Is this Insert command executed successful?**
- ❑ INSERT INTO CANDIDATE VALUES('MIT1021', 'Raghu', 'COMP.SC');
- ❑ If user enters Branch as –'Comp.Sc' , it is **rejected** with constraint error message.

* We will see other inbuilt functions later

Exercise

Create a table **Student**(Regno,Name,Mark1,Mark2), Regno is primary key and **Mark1** must accept values in the range **0** to **100** , **Mark2** must accept values in the range **0** to **150**.

```
CREATE TABLE Student(Regno Number(9) PRIMARY KEY , Name Varchar2(10), Mark1  
Number(3) CHECK (Mark1>=0 AND Mark1<=100),Mark2 Number(3) CHECK( Mark2  
BETWEEN 0 AND 150 ));
```

What is the result of following insert command ?

```
INSERT INTO Student VALUES (100, 'Raghu',99, 159)
```

Error - ORA-02290: check constraint (SYSTEM.SYS_C007566) violated

Exercise

Create a table **Student**(Regno, Name, Grade), Regno is primary key and Grade must accept only values- A+,A,B,C,D,E,F

```
CREATE TABLE Student(Regno Number(9) PRIMARY KEY , Name  
Varchar2(10), Grade Char(2) CHECK ( Grade IN ('A+','A','B','C','D','E','F')) ) ;
```

What is the result of following insert command ?

```
INSERT INTO Student VALUES(100, 'Raghu', 'B+')
```

Error - ORA-02290: check constraint (SYSTEM.SYS_C007563) violated

Exercise

Create a table **Course_Structure**(Subject_id, Sub_Name, Credits, Sem), Subject_id is primary key and **Subject Id** must **start with MCA**. Credit values must be 1 ,2,3,4. **E.g.** MCA4151, MCA5151 etc.

```
CREATE TABLE Course_Structure (Subject_id Varchar2(10) PRIMARY KEY  
CHECK(Subject_id LIKE 'MCA%'), Sub_name Varchar2(20), Credits Number(1)  
CHECK ( Credits IN (1,2,3,4))) ;
```

What is the result of following insert command ?

```
INSERT INTO Course_structure VALUES('MCA100', 'DBMS', 4)
```

Error - ORA-02290: check constraint (SYSTEM.SYS_C007557) violated

DEFAULT

The DEFAULT constraint is used to provide a default value for a column.

Example:

```
CREATE TABLE Persons (  
  ID Number(3) NOT NULL,  
  LastName varchar(10) NOT NULL, FirstName varchar(10),  
  Age Number(2), City varchar(15) DEFAULT 'Manipal' );
```

Example:

```
INSERT INTO Persons(ID,Lastname) VALUES(100,'AAA');
```

Inserts value to ID=100 , Lastname=AAA , **FirstName= NULL** , **Age=NULL** & City takes value **Manipal** automatically assigned though City value is not specified in the INSERT command.

Naming the Constraints

- If user do not specifies Constraint Name while defining Constraints, System itself gives a name. System uses auto generate method to give unique constraints names such as – **SYS_C0003461** etc. As constraint names have to be unique. In case of constraint violation, it is easy to user to track the constraint if user defined constraint name is given.
- Use **CONSTRAINT *name_of_constraint*** along with constraint definition in CREATE or ALTER table.
- Create following tables with constraint names.

Department

Attribute	Constraint	Constr_Name
Dname	PRIMARY KEY	Dname_PK
	Refers Organization	fk_Orga
Course_Type	Check	Chk_Type
Numb_of_Sem		
In_take_stud		
Dep_Phone	Not Null	NoNul
	Unique	Unq_Ph

Organization

Attribute	Constraint	Constr_Name
Dept_name	Primary Key	dp_PK
Head		

Example

- CREATE TABLE Organization(Dept_name varchar2(8) CONSTRAINT dp_PK PRIMARY KEY, Head varchar2(10));
- CREATE TABLE Department (Dname varchar2(8) CONSTRAINT Dname_PK PRIMARY KEY CONSTRAINT fk_Orga REFERENCES Organization, Course_Type varchar2(8) CONSTRAINT Chk_Type CHECK(Course_Type IN('MCA','MTech ', 'BTech', 'MS')), Numb_of_Sem Number(1), In_take_stud Number(2), Dep_Phone Number(10) CONSTRAINT noNul NOT NULL CONSTRAINT Unq_Ph UNIQUE);

Exercise

Create a table **Course_Structure**(Subject_id, Sub_Name, Credits, Sem), Subject_id is primary key and **Subject Id** must **start with MCA**. Credit values must be 1 ,2,3,4. **E.g.** MCA4151, MCA5151 etc. Assign proper constraint names.

```
CREATE TABLE Course_Structure (Subject_id Varchar2(10) CONSTRAINT SubID_PK PRIMARY  
KEY CONSTRAINT Starts_MCA CHECK(Subject_id LIKE 'MCA%'), Sub_name Varchar2(20),  
Credits Number(1) CONSTRAINT Credt_Range CHECK ( Credits IN (1,2,3,4))) ;
```

What is the result of following insert command ?

```
INSERT INTO Course_structure VALUES('MCA100', 'DBMS', 4);
```

Error - ORA-02290: check constraint (SYSTEM.STARTS_MCA) violated

Exercise

Create following tables(DEPT & EMP) with given constraint names.

DEPT	Attribute	Data Type	Size	Constraints	Constraint Name
	DNO	VARCHAR2	2	PRIMARY KEY	
	DNAME	VARCHAR2	10		
	HEAD_OFFC_CITY	VARCHAR2	10	UDP,BNG,HYD,MUB,LA	Valid_offc_city

EMP	Attribute	Data Type	Size	Constraints	Constraint Name
	EMPNO	NUMBER	3	PRIMARY KEY	PK_Empno
	ENAME	VARCHAR2	10		
	MGRNO	NUMBER	3	References EMP(EMPNO)	FK_MgrNo_EMP
	DEPTNO	VARCHAR2	2	References DEPT(DNO)	FK_Deptno_DEPT
	DOB	DATE			
	DOJ	DATE		DOJ>DOB	DOJ_Grtr_DOB
	SAL	NUMBER	7,2	SAL>30000	SAL_Grtr_30K

CREATE TABLE ... AS SELECT...

The **CREATE TABLE ... AS SELECT...** statement is used to create a new table having same/partial structure of an existing table given with SELECT statement.

EMP_SPOUSE

Attribute	Data Type	Size
EMPNO	NUMBER	3
ENAME	VARCHAR2	10
SPOUSE_NAME	VARCHAR2	10

We can create EMP_SPOUSE table by copying structure for EMPNO and ENAME from EMP table.

EMP

Attribute	Data Type	Size	Constraints	Constraint Name
EMPNO	NUMBER	3	PRIMARY KEY	PK_Empno
ENAME	VARCHAR2	10		
MGRNO	NUMBER	3	References EMP(EMPNO)	FK_MgrNo_EMP
DEPTNO	VARCHAR2	2	References DEPT(DNO)	FK_Deptno_DEPT
DOB	DATE			
DOJ	DATE		DOJ>DOB	DOJ_Grtr_DOB
SAL	NUMBER	7,2	SAL>30000	SAL_Grtr_30K

CREATE TABLE ... AS SELECT...

CREATE TABLE EMP_SPOUSE(ENO,NAME) AS SELECT EMPNO,ENAME FROM EMP;

ALTER TABLE EMP_SPOUSE ADD(Spouse_name Varchar2(10));

EMP_SPOUSE

Attribute	Data Type	Size
ENO	NUMBER	3
NAME	VARCHAR2	10
SPOUSE_NAME	VARCHAR2	10

EXAMPLE: Create a table EMP_SPOUSE using already existing EMP table

*More about ALTER TABLE we will see in coming slides

EMP

Attribute	Data Type	Size	Constraints	Constraint Name
EMPNO	NUMBER	3	PRIMARY KEY	PK_Empno
ENAME	VARCHAR2	10		
MGRNO	NUMBER	3	References EMP(EMPNO)	FK_MgrNo_EMP
DEPTNO	VARCHAR2	2	References DEPT(DNO)	FK_Deptno_DEPT
DOB	DATE			
DOJ	DATE		DOJ>DOB	DOJ_Grtr_DOB
SAL	NUMBER	7,2	SAL>30000	SAL_Grtr_30K

Drop Table Constructs

The **DROP TABLE** statement allows you to remove or delete a table from the database.

Syntax:

```
DROP TABLE tablename;
```

Example: DROP TABLE Emp;

Alter Table Constructs...

The **ALTER TABLE** statement is used to **add, modify, or drop/delete columns/constraints** in a table.

The SQL ALTER TABLE statement is also used to rename a table.

Adding Column

Syntax:

```
ALTER TABLE table_name ADD (column_name1 column-  
definition,      column_name1 column-definition,.....) ;
```

Example: Add column Salary and Phone to Emp table(already existing).

```
ALTER TABLE Emp ADD (Salary Number(7), Phone Number(10));
```

..Alter Table Constructs

Modifying Column

Syntax:

```
ALTER TABLE table_name  
MODIFY (column_1 column_type, column_2 column_type, ... column_n  
        column_type);
```

Example: Increase the size of Salary column & modify Name column definition by adding NOT NULL rule

```
ALTER TABLE Emp MODIFY ( Name VARCHAR(25) NOT NULL, Salary  
Number(9,2) );
```

..Alter Table Constructs

DROP a Column

Syntax:

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

Example: Drop a column EName from Emp table.

```
ALTER TABLE Emp DROP COLUMN EName;
```

..Alter Table Constructs

RENAME a Table

Syntax:

```
ALTER TABLE table_name RENAME TO New_table_name;
```

Example: Add column Salary to Emp table.

```
ALTER TABLE Emp RENAME TO Employee;
```

..Alter Table Constructs

Adding CHECK Constraint to a column

Syntax:

```
ALTER TABLE table_name  
    ADD CONSTRAINT constraint_name CHECK( p );
```

Where **p** - predicate

Example: Add constraint to **Students** table to check **mark2** column takes values only in the **range 0 to 100**.

```
ALTER TABLE Students  
    ADD CONSTRAINT check_mark_range  
    CHECK (mark2>=0 AND mark2<=100);
```

..Alter Table Constructs

Adding UNIQUE Constraint to a column

Syntax:

```
ALTER TABLE table_name
```

```
ADD CONSTRAINT constraint_name UNIQUE( column1,column2,..columnn ) );
```

Example: Add constraint to **Students** table make **Phone** column as Unique.

```
ALTER TABLE Student
```

```
ADD CONSTRAINT uniq_phone
```

```
UNIQUE(Phone);
```


..Alter Table Constructs

Adding PRIMARY KEY Constraint to a column

Syntax:

```
ALTER TABLE table_name
```

```
ADD CONSTRAINT constraint_name
```

```
PRIMARY KEY (column1, column2, ... column_n) ;
```

Example: Assume that Person(Fname, Lname, Address) table is already created. Add constraint to **Person** table to make (**FName,LName**) column as Primary Key.

```
ALTER TABLE Person ADD CONSTRAINT F_L_Name_FK
```

```
PRIMARY KEY (FName,LName);
```

..Alter Table Constructs

Adding FOREIGN KEY Constraint to a column

Syntax:

```
ALTER TABLE table_name
```

```
ADD CONSTRAINT constraint_name
```

```
FOREIGN KEY (column1, column2, ... column_n)
```

```
REFERENCES parent_table (column1, column2, ... column_n);
```

Example: Assume that **Person(Fname, Lname, Address)** table is already created with **(Fname,LName)** as **Primary Key**. Also a table **Customer(Cust_Id, Cust_FName,Cust_Lname,Credits)** is also created already.

Now we want to make (Cust FName,Cust Lname) as foreign key referencing **Person**

```
ALTER TABLE Customer ADD CONSTRAINT Cust_FLName_FK
```

```
FOREIGN KEY(Cust_Fname , Cust_Lname) REFERENCES Person;
```

..Alter Table Constructs

Removing Constraints

Syntax:

```
ALTER TABLE table_name
```

```
DROP CONSTRAINT constraint_name ;
```

Example: Assume that **Person(Fname, Lname, Address)** table is already created with (Fname,LName) as **Primary Key**. Also a table **Customer(Cust_Id, Cust_FName,Cust_Lname,Credits)** is also created already. Now we want to remove foreign key constraint from (Cust_FName,Cust_Lname).

```
ALTER TABLE Customer DROP CONSTRAINT Cust_FLName_FK;
```

INSERT

Inserts a new record at the end of given table.

Syntax-

INSERT INTO table_name VALUES (value1,value2,....)

Example: Insert a record to a table Course(Course_id,title,Dept_Name, Credits)

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

There will be **1 to 1 mapping** between **values** given and order in which **columns** are created in relation Course.

1st value '**CS-437**' is mapped to column Course_id,

2nd value '**Database Systems**' is mapped to column **title** and so on.

..INSERT

Syntax-

INSERT INTO table_name(column1,column2,..) VALUES (value1,value2,....)

Example: Insert a record into Course table having values to Course_id, Dept_Name columns only. Course(Course_id,title,Dept_Name,Credits)

insert into *course* (*course_id,dept_name*) **values** ('CS-438', 'Comp. Sci.');

It is equivalent to –

insert into *course* **values** ('CS-438', NULL, 'Comp. Sci.', **NULL**);

Note: **NULL** is not same as **'NULL'**

Insert into... Select .. From...

- Some time instead of giving data for every tuple in the INSERT INTO command, we can insert tuples on the basis of the result of a query.
- Using SELECT statement as sub query in the INSERT INTO, we can select (copy) some set of records from a relation(source) and insert into another relation(Destination).
- Note that we need to take care of datatype and size compatibility.

STUD	Rollno	Name	Course	Dept
	101	Ajit	Algorithms	CS
	102	Ravi	IoT	IT
	103	Anish	Algorithms	MCA
	101	Ram	ML	MCA

MARKS	Rno	Course	Marks	Attendance

Example: Insert Rollno and course information of students enrolled to MCA department into MARKS relation.

```
INSERT INTO MARKS(RNo, Course) SELECT Rollno, Course FROM STUD WHERE Dept='MCA';
```

..INSERT

Syntax-

INSERT INTO table1(column1,column2,..) **SELECT** column1,column2,.. **FROM** table2;

- **Example:** Consider the tables **Student**(Id, Name, D_name, tot_cred) and **Instructor**(Id, Name, Dept_name, Salary). Add all instructors to the *student* relation with tot_creds set to 0

```
insert into student
select Id, Name, Dept_name, 0
from instructor;
```

OR

```
insert into student(ID, name, D_name)
select ID, name, dept_name
from instructor;
```

The **select from where** statement is evaluated fully before any of its results are inserted into the relation

..INSERT (date value)

Example: Assume a table Stud (Rno, Name, Birth_Date)

- Insert a record into STUD table.

```
INSERT INTO Stud VALUES(19011102, 'Ajay', TO_DATE('21-09-2001','DD-MM-YYYY'));
```

- TO_DATE () is a oracle inbuilt function, which converts given date value (in the form character value) into date type.
- Date has a default format set. Example: Default format is say : DD-MON-YY , then you can enter data as below without TO_DATE()

```
INSERT INTO Stud VALUES(19011103, 'Aman', '21-OCT-2001');
```


UPDATE

To modify any column/s value in a already existing record.

Syntax:

```
UPDATE table_name SET column1=value1,column2=value2,...  
WHERE condition involving any of column/s in the table ;
```

Example: Consider the table Instructor(Id, Name, Dept_name, Salary).
Increase the salary of instructor with ID I201 by 10%.

```
UPDATE Instructor SET Salary=Salary+Salary*0.1 WHERE Id='I201';
```

..UPDATE

- **Example:** Consider the table **Instructor**(Id, Name, Dept_name, Salary). Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others receive a 5% raise

- Write two update statements:

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

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

- The order is important

..UPDATE –using CASE

- Same query(previous slide) as before but with case statement

update *instructor*

set *salary* = **case**

when *salary* <= 100000 **then** *salary* * 1.05

else *salary* * 1.03

end;

Assume the table Emp(Empno,ename,deptnosal)

update emp set sal=**case**

when sal<=3000 **then** sal*1.1

when sal<=5000 **then** sal*1.05

else sal*1

end;

DELETE

Syntax:

DELETE FROM table_name WHERE condition;

Example:

- Delete all instructors

delete from *instructor*

- Delete all instructors from the Finance department

delete from *instructor*

where *dept_name*= 'Finance';

..DELETE

Syntax:

DELETE FROM *table_name* WHERE *condition*;

Note- Condition is involving some sub-query

Example:

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the 'Watson' building.

```
delete from instructor  
where dept_name in (select dept_name  
                        from department  
                        where building = 'Watson');
```

..DELETE

Example:

Delete all instructors whose salary is less than the average salary of instructors

delete from *instructor*

where *salary* < (**select avg** (*salary*) **from** *instructor*);

- Problem: as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
 1. First, compute **avg** salary and find all tuples to delete
 2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

END