

## GENERAL CONSTRAINTS- PRIMARY AND FOREIGN KEY

### TYPES OF CONSTRAINTS

Primary Key Constraint

Foreign Key Constraint

Unique Constraint

Check Constraint

Not Null Constraint

### Primary Key vs Foreign Key

Primary keys serve as unique identifiers for each row in a database table.

Foreign keys link data in one table to the data in another table. A foreign key column in a table points to a column with unique values in another table (often the primary key column) to create a way of cross-referencing the two tables.

### CREATE TABLE EXAMPLE

Customer\_Account\_Details

Column Name	Data type	Constraints
Cust_ID	Number(5)	Primary key of the table
Cust_Last_Name	Varchar2(20)	
Cust_Mid_Name	Char(3)	
Cust_First_Name	Varchar2(20)	
Account_No	Number(4)	
Account_Type	Varchar2(15)	
Bank_Branch	Varchar2(20)	
Cust_Email	Varchar2(30)	

### SQL - CREATE TABLE

## Implementing PRIMARY KEY ,NOT NULL and UNIQUE

EXAMPLE :

```
CREATE TABLE Customer_Account_Details( Cust_ID Number(5) CONSTRAINT
Cust_Pkey Primary Key, Cust_Last_Name VarChar2(20), Cust_Mid_Name Char(3),
Cust_First_Name VarChar2(20), Account_No Number(4) , Account_Type
Varchar2(15), Bank_branch Varchar2(20), Cust_Email VarChar2(30));
```

### Implementation of Primary Key and Foreign Key Constraints Customer\_Loan

Column Name	Data type	Constraints
LoanNo	Number(4)	It is the primary key of the table
Cust_ID	Number(5)	It can take only those values which are present in the Cust_ID of the Customer_Account_Details Table
Amount_In_dollar	Number(6,2)	

### Implementing Foreign Key

EXAMPLE :

```
CREATE TABLE Customer_Loan(
LoanNo Number(4) CONSTRAINT Loan_PKey PRIMARY KEY, Cust_ID Number(5)
CONSTRAINT CustID_FKey REFERENCES Customer_Account_Details(Cust_ID),
Amount_In_Dollar Number(6,2) );
```

### Implementation of Self Referencing Foreign key Employee\_Details

Column Name	Data type	Constraints
-------------	-----------	-------------

Employee_ID	Number(6)	Primary key of the table
Employee_Last_Name	Varchar2(20)	
Employee_Mid_Name	Varchar2(3)	
Employee_First_Name	Varchar2(20)	
Employee_Email	Varchar2(30)	
Employee_Dept	Number(2)	Default 'HR'
Manager_ID	Varchar2(30)	It can take only those values which are present in Employee_ID column

### Implementing Self Referential Foreign Key and Default

#### EXAMPLE :

```
CREATE TABLE Employee_Details( Employee_ID Number(5) CONSTRAINT
Employee_PKey PRIMARY KEY, Employee_Last_Name Varchar2(20),
Employee_Mid_Name Char(3), Employee_First_Name Varchar2(20),
Employee_Email Varchar2(30), Department Varchar2(10) default 'HR',
Manager_ID Number(5) CONSTRAINT Manager_FKey REFERENCES
Employee_Details(Employee_ID));
```

#### Constraints- Check, Not Null and Unique Constraints

General syntax for adding a CHECK constraint to a column when creating a new table:

```
CREATE TABLE table_name
```

```
( column1 datatype [CONSTRAINT constraint_name] CHECK (condition), column2
datatype,... );
```

#### Customer\_FixedDeposit

Column Name	Data type	Constraints
-------------	-----------	-------------

FixedDeposit_No	Number(4)	It is the primary key of the table
Cust_ID	Number(5)	It can take only those values which are present in the Cust_ID of the Customer_Account_Details Table
Amount_In_dollar	Number(6,2)	
Rate_Of_Intrest	Number(3,1)	It can take values only between 2.5 to 12.0

### Implementing Check constraint

#### EXAMPLE :

```
CREATE TABLE Customer_FixedDeposit( FixedDeposit_ID Number(4)
CONSTRAINT FixedDeposit_PKey PRIMARY KEY, Cust_ID Number(5)
CONSTRAINT FixedDeposit_FKey REFERENCES
Customer_Account_Details(Cust_ID), Amount_In_Dollar Number(6,2),
Rate_Of_Interest Number(3,1) CONSTRAINT Intrest_Ccheck
Check(Rate_of_Interest >=2.5 and Rate_Of_Interest<=12.0) );
```

#### Not Null & Unique Constraints – Syntax

```
CREATE TABLE table_name ( column1 datatype NOT NULL, column2 datatype NOT
NULL UNIQUE, ... );
```

#### Customer\_Account\_Details

Column Name	Data type	Constraints
Cust_ID	Number(5)	Primary key of the table
Cust_Last_Name	Varchar2(20)	Not Null column
Cust_Mid_Name	Char(3)	

Cust_First_Name	Varchar2(20)	
Account_No	Number(4)	Unique column
Account_Type	Varchar2(15)	
Bank_Branch	Varchar2(20)	
Cust_Email	Varchar2(30)	Unique column

### CREATE TABLE- Not Null, Unique

#### Implementing NOT NULL and UNIQUE

EXAMPLE :

```
CREATE TABLE Customer_Account_Details(Cust_ID Number(5) CONSTRAINT
Cust_Pkey Primary Key, Cust_Last_Name VarChar2(20) CONSTRAINT
CustLastName_Nnull NOT NULL, Cust_Mid_Name Char(3), Cust_First_Name
VarChar2(20), Account_No Number(4) CONSTRAINT Cust_Account_Unq UNIQUE,
Account_Type Varchar2(15), Bank_branch Varchar2(20), Cust_Email VarChar2(30)
CONSTRAINT CustEmail_Unq UNIQUE );
```

### Constraints- Composite Primary and Foreign Key & Alter

#### Composite Primary Key Constraints

Customer\_Transaction

Column Name	Data type	Constraints
Cust_ID	Number(5)	It can take only those values which are present in Cust_ID column of Customer_Account_Details Table
Transaction_Date	Date	(Cust_ID,Transaction_Date) are together forming the Primary Key of the table.

Transaction_Type	Varchar2(20)	
Transaction_Amt_InDollar	Number(6,2)	
Total_Available_Balance_ InDollar	Number(7,2)	

### Implementing Composite Primary key constraint

#### EXAMPLE :

```
CREATE TABLE Customer_Transaction(Cust_ID Number(5) Constraint
CustTran_Fkey references Customer_Account_Details(Cust_ID), Transaction_Date
Date, Transaction_Type Varchar2(20), Transaction_Amt_InDolllar Number(6,2),
Total_Available_Balance_InDollar Number(7,2), CONSTRAINT
Customer_Transaction_PKey PRIMARY KEY(Cust_ID,Transaction_Date)
```

### Implementing Composite Foreign key constraint

#### EXAMPLE :

```
CREATE TABLE Feedback( TrainerID Number(2), BatchName Varchar2(10),
CourseID Varchar2(10), FBRate Number(3,2), CONSTRAINT
Feedback_BatchSchedule_FKey FOREIGN KEY(BatchName,CourseID)
REFERENCES Batch_Schedule(BatchName,CourseID) );
```

### Add/Drop/Modify Column

Syntax:

```
ALTER TABLE tablename (ADD/MODIFY/DROP column_name)
```

```
ALTER TABLE Customer_Account_Details
```

```
ADD Contact_Phone Char(10);
```

```
ALTER TABLE Customer_Account_Details MODIFY Contact_Phone Char(12);
```

```
ALTER TABLE Customer_Account_Details DROP (Contact_Phone);
```

- Used to modify the structure of a table by adding and removing columns
- The ALTER TABLE statement with MODIFY option cannot be used to change the name of a column or table.
- Column to be modified should be empty to decrease column length

- Column to be modified should be empty to change the data type
- If the table has only one column, the ALTER TABLE statement cannot be used to drop that column because that would render the table definition invalid.

### Add/Drop Constraint

ALTER TABLE Customer\_Account\_Details ADD CONSTRAINT Pkey1 PRIMARY KEY (Account\_No);

ALTER TABLE Customer\_Account\_Details ADD CONSTRAINT Pkey2 PRIMARY KEY (Account\_No, Cust\_ID);

ALTER TABLE Customer\_Account\_Details DROP PRIMARY KEY;

Or

ALTER TABLE Customer\_Account\_Details DROP CONSTRAINT Pkey1;

ALTER TABLE Customer\_Transaction (ADD CONSTRAINT Fkey1 FOREIGN KEY (Cust\_ID) REFERENCES Customer\_Account\_Details (Cust\_ID));

ALTER TABLE Customer\_Transaction DROP CONSTRAINT Fkey1;

- A table can have one or more Foreign keys
- Adding a foreign key constraint using ALTER TABLE command will result in an error if the existing data in master or child table does not support the foreign key restriction.
- ALTER TABLE statement can be used to Add or Drop **primary key** constraint to / from a table
- ALTER TABLE statement can be used to Add or Drop **foreign key** constraint to / from a table
- ALTER TABLE statement can be used to Add or Drop **Unique** constraint to / from a table
- ALTER TABLE statement can be used to Add or Drop **check** constraint to / from a table
- If a table already has a primary key, then adding a primary key using the ALTER TABLE statement results in an error.
- The RDBMS will not allow a PRIMARY KEY constraint (using the ALTER TABLE statement) on column(s) if the column(s) has NULL or duplicate values

### SQL - DROP TABLE

#### DROP TABLE

- ## Truncate Table

- ```
TRUNCATE TABLE Customer_Account_Details;
```

## DML Commands

There are four commands in DML:

- ## INSERT COMMAND

- ## SQL-DML- INSERT

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode,Country)VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

## INSERTING NULL IN TO THE TABLE

```
insert into InsertNullDemo values(101,'Mike',NULL);
```



## SQL - DELETE FROM

With or without WHERE clause

Syntax: DELETE FROM *tablename* WHERE *condition*

### Deleting All Rows

DELETE FROM Customer\_Account\_Details;

## SQL - DELETE FROM

### Deleting Specific Rows

DELETE FROM Customer\_Account\_Details WHERE Cust\_ID = 102;

### Delete Vs Truncate

| DELETE                                                                  | TRUNCATE                                                          |
|-------------------------------------------------------------------------|-------------------------------------------------------------------|
| Data can be recovered                                                   | Data cannot be recovered.                                         |
| DML statement                                                           | DDL statement                                                     |
| DELETE does not release the memory occupied by the records of the table | TRUNCATE releases the memory occupied by the records of the table |

## SQL-UPDATE

Syntax:

UPDATE *tablename* SET *column\_name* = *value* [ WHERE *condition* ]

### Updating All Rows

UPDATE Customer\_FixedDeposit  
SET Rate\_of\_Interest = NULL;

### Updating Particular rows

UPDATE Customer\_FixedDeposit SET Rate\_of\_Interest = 7.3 WHERE  
Amount\_in\_Dollar > 3000;

### Updating Multiple Columns

```
UPDATE Customer_FixedDeposit SET Amount_In_Dollar = 100.0 , Rate_of_Interest = 7.3 WHERE Cust_ID = 104;
```

### **SQL DML- SELECT**

**To select set of column names,**

```
SELECT column1, column2,... FROM TableName
```

#### **Example**

```
SELECT * FROM Customer_Account_Details;
```

Or

```
SELECT Cust_ID, Cust_Last_Name, Cust_First_Name, CustEmail FROM Customer_Account_Details;
```

#### **Get all Customers Name:**

```
SELECT ALL Cust_Last_Name FROM Customer_Account_Details;
```

Or

```
SELECT Cust_Last_Name FROM Customer_Account_Details;
```

#### **Get all distinct Customer Name**

```
SELECT DISTINCT Cust_Last_Name FROM Customer_Account_Details;
```

#### ***List all customers with an account balance > \$10000***

```
SELECT Cust_ID, Total_Available_Balance_in_Dollars FROM Customer_Transaction WHERE Total_Available_Balance_in_Dollars > 10000.00;
```

### **Simple Examples- Querying Relational Data**

#### ***List all customers with an account balance > \$10000***

```
SELECT Cust_ID, Total_Available_Balance_in_Dollars FROM Customer_Transaction WHERE Total_Available_Balance_in_Dollars > 10000.00;
```

### **SQL DML- SELECT, Where clause with operators**

#### ***List the Cust\_ID, Account\_No of 'Graham'***

```
SELECT Cust_ID, Account_No FROM Customer_Account_Details WHERE Cust_First_Name = 'Graham';
```

#### ***List all Cust\_ID where Total\_Available\_Balance\_in\_Dollars is atleast \$10000.00***

```
SELECT Cust_ID FROM Customer_Transaction WHERE Total_Available_Balance_in_Dollars >= 10000.00;
```

#### ***List all Cust\_ID, Cust\_Last\_Name where Account\_type is 'Savings' and Bank\_Branch is 'Capital Bank'***

SELECT Cust\_ID, Cust\_Last\_Name FROM Customer\_Account\_Details WHERE Account\_Type = 'Savings' AND Bank\_Branch = 'Capital Bank';

**List all Cust\_ID, Cust\_Last\_Name where neither Account\_type is 'Savings' and nor Bank\_Branch is 'Capital Bank'**

SELECT Cust\_ID, Cust\_Last\_Name FROM Customer\_Account\_Details WHERE NOT Account\_Type = 'Savings' AND NOT Bank\_Branch = 'Capital Bank';

**List all Cust\_ID with balance in the range \$10000.00 to \$20000.00.**

SELECT Cust\_ID FROM Customer\_Transaction WHERE Total\_Available\_Balance\_in\_Dollars >= 10000.00 AND Total\_Available\_Balance\_in\_Dollars <= 20000.00;

Or

SELECT Cust\_ID FROM Customer\_Transaction WHERE Total\_Available\_Balance\_in\_Dollars BETWEEN 10000.00 AND 20000.00;

**List all customers who have account in Capital Bank or Indus Bank.**

SELECT Cust\_ID FROM Customer\_Account\_Details WHERE Bank\_Branch = 'Capital Bank' OR Bank\_Branch = 'Indus Bank';

Or

SELECT Cust\_ID FROM Customer\_Account\_Details WHERE Bank\_Branch IN ('Capital Bank', 'Indus Bank');

**List all Accounts where the Bank\_Branch begins with a 'C' and has 'a' as the second character**

SELECT Cust\_ID, Cust\_Last\_Name, Account\_No FROM Customer\_Account\_Details WHERE Bank\_Branch LIKE 'Ca%';

**List all Accounts where the Bank\_Branch column has 'a' as the second character.**

SELECT Cust\_ID, Cust\_Last\_Name, Account\_No FROM Customer\_Account\_Details WHERE Bank\_Branch LIKE '\_a%';

**List employees who have not been assigned a Manager yet.**

SELECT Employee\_ID FROM Employee\_Manager WHERE Manager\_ID IS NULL;

**List employees who have been assigned to some Manager.**

SELECT Employee\_ID FROM Employee\_Manager WHERE Manager\_ID IS NOT NULL;

**List the Cust\_ID and their account balances, in the increasing order of the balance**

```
SELECT Cust_ID, Total_Available_Balance_in_Dollars FROM Customer_Transaction
ORDER BY Total_Available_Balance_in_Dollars;
```

### Aggregate Functions

- Used when information you want to extract from a table has to do with the data in the entire table taken as a set.
- Aggregate functions are used in place of column names in the SELECT statement
- The aggregate functions in sql are :
- SUM( ) , AVG( ) , MAX( ) , MIN( ) , COUNT( )

```
SUM ( [ DISTINCT ] column-name / expression )
AVG ( [ DISTINCT ] column-name / expression )
MIN ( expression )
MAX ( expression )
COUNT ( [ DISTINCT ] column-name )
COUNT ( * )
```

### Aggregate Functions- MIN

- Returns the smallest value that occurs in the specified column
- Column need not be numeric type

*List the minimum account balance.*

```
SELECT MIN (Total_Available_Balance_in_Dollars) FROM Customer_Transaction;
```

### Aggregate Functions- MAX

- Returns the largest value that occurs in the specified column
- Column need not be numeric type

Example:

```
SELECT MAX (Total_Available_Balance_in_Dollars) FROM Customer_Transaction;
```

### Aggregate Functions- AVG

- Returns the average of all the values in the specified column
- Column must be numeric data type

Example:

- *List the average account balance of customers.*

```
SELECT AVG (Total_Available_Balance_in_Dollars) FROM Customer_Transaction;
```

### **Aggregate Functions- MIN, SUM**

- Adds up the values in the specified column
- Column must be numeric data type
- Value of the sum must be within the range of that data type

**Example:**

*List the minimum and Sum of all account balance.*

```
SELECT      MIN      (Total_Available_Balance_in_Dollars),      SUM
(Total_Available_Balance_in_Dollars) FROM Customer_Transaction;
```

### **Aggregate Functions- COUNT**

- Returns the number of rows in the table

*List total number of Employees.*

```
SELECT COUNT (*) FROM Employee_Manager;
```

*List total number of Employees who have been assigned a Manager.*

```
SELECT COUNT (Manager_ID) FROM Employee_Manager;
```

Count(\*) = No of rows, regardless of NULLs

Count(Column Name) = No. of rows that do not have NULL Value

*List total number of account holders in the 'Capital Bank' Branch.*

```
SELECT COUNT (*) FROM Customer_Details WHERE Bank_Branch = 'Capital
Bank';
```

*List total number of unique Customer Last Names.*

```
SELECT COUNT (DISTINCT Cust_Last_Name) FROM Customer_Details;
```

### **Aggregate Functions**

- Create, Alter and Drop are the DDL commands
- Update, Insert into, Delete from, are the basic DML commands that add or remove data from tables
- Select statement in its various flavours is used to retrieve information from the table
- Aggregate functions work on all the rows of the table taken as a group (based on some condition optionally)