**Chapter 1🡨Database**

A D.B consists of an organized collection of data. There are various models that are used to organize data such as:

* Hierarchical model
  + organizes data in a tree structure
  + hierarchy of parent and child data segments
* Network model
  + for more than one parent to be associated with a child
* Relational model
  + Our focus lies here
* Object Relational model
  + provide a middle ground between relational databases and object-oriented databases
* Object Oriented Relational model.
  + info is represented in the form of objects as used in object-oriented programming

**RELATIONAL MODEL**

matches data by using common characteristics found within the data set. Resulting goups of data are organized and are much easier to understand

**Data hierarchy**: ordering of data types by size

* **Field**: group of characters forming a single data item
  + example, “John”
* **Record, row, tuple**: a group of related fields
  + An individual’s record containing ssn, lastname, firstname, city, state and zip
* **Column**: is a set of data of data values of a particular type. Field value is used to refer specifically to the single item that exists at the intersection between one row and one column such as “Title”
* **Table, Entity**: a group of related records
  + Table “Customer” contains all the information about the various experiments
* **Database**: collection of related files, called tables.

**SQL🡨**Structured Query language

* Data Definition Language(DDL) are used to define the D.B structure or schema
  + Create, alter, drop,truncate…
* Data Manipulation Language(DML) statements are used for managing data within schema objects. Some examples
  + Select,insert,update,delete
* Data Control Language (DCL)
  + Grant, Revoke
* Transaction Control (TCL)statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.
  + Commit,savepoint,rollback

**Chapter 2 (Database Design)**

**Normalization**

The process of refining tables, keys, columns, and relationships to create an efficient database.Normalization usually involves:

* Dividing a database into two or more tables
* Defining relationships between the tables

Advantages

* Reduce duplication of data
* Avoiding irregularities. Irregularities which can cause insert, update and delete issues.

**2.3 Normal Forms**

First normal form

* Each row and column intersection must contain one and only one value. Must be atomic. Eliminate repeating groups.

Second normal form

* Every non-key column must depend on the entire primary key. Eliminate partial key dependencies.

Third normal form

* No non-key column depends on another non-key column. Eliminate transitive dependencies.

Fourth normal form

* Forbids independent relationships between primary key columns and non-key columns.

Fifth normal form

* Breaks tables into the smallest possible pieces in order to eliminate redundancy

**2.4 Primary key, Foreign Key, Candidate or Alternate Key**

**Primary Key**

* ensures that no two records have same value for that field
* A field whose values are unique for each record in a table
* Creates relationship between tables
* May be composed of one or multiple columns
  + Called a Compound or a composite primary key key

**Unique/Candidate/Alternate key**

* A candidate key is a combination of attributes that can be uniquely used to identify a database record without any extraneous data. Each table may have one or more candidate keys. One of these candidate keys is selected as the table primary key.
* Alternate key used strictly for data retrieval purposes.
* May be composed of one or multiple columns
  + Called a Compound or a composite candidate key

**Foreign key**

* A key field that identifies records in a different table The foreign key is used to establish a relationship with another table or tables.

2.8 Entity Relationship Diagrams (ER)

* One-to-one
* One-to-many

**Chapter 3 (DDL Commands)**

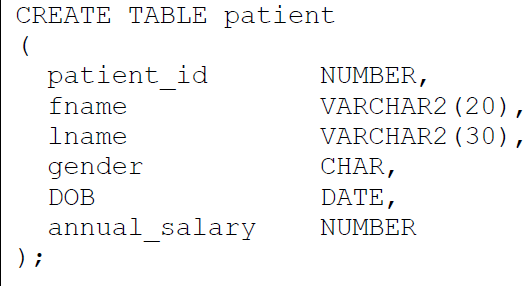
Things to note:

* For readability, upper case all the Oracle Reserved words.
* Oracle is not case sensitive. The data being inserted is case sensitive.
* Separate every column definition with a comma except for the last one
* Use a semicolon to end the SQL statement.
* Do not put any blank lines between your code otherwise you will get an error message.
* two tables cannot have same name.
* Identifiers cannot be more than 30 characters long.
* A table can have up to 1000 columns

Some common types:

* VARCHAR2(n): character data where **n** represents the maximum size=4000
* CHAR(n) Fixed-length character length 1 and the maximum size is 2000.
* NUMBER(p,s) Numeric column where **p** indicates precision(total number of digits to the left and
* right of the decimal position- max 38 digits) and **s** indicates scale (numbe of positions
* to the right of the decimal.
* DATE Stores date and tiime between January 1, 4712 BC and December 31, 9999 AD.
* Oracle’s default date format is DD-MON-YY.

***Example 3.1a (Create table)***



***Example 3.1b (DESC command)***

DESC provides information about the columns in a table. Semicolon not required at the end

DESC patient;

***Example 3.1c (View contents of table)***

SELECT \* FROM patient;

***Example 3.1d (Drop table command)***

Moves table to recycle bin. All data is erased

Drop TABLE patient;

*Example 3.1e (Flashback)*

*A dropped table can be recovered using Flashback*

FLASHBACK TABLE patient TO BEFORE DROP;

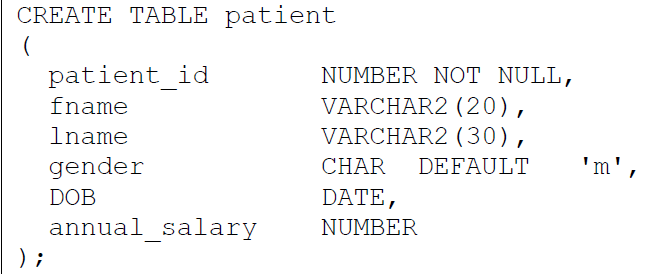
***Example 3.1f (Recyclebin)***

Purging from the recyclebin

PURGE RECYCLEBIN;

***Example 3.1g (Not NULL and default constraints)***

Create a table with a NOT NULL and DEFAULT constraint. If no value is entered for a column, the value is considered NULL, indicating an absence of data.



***Example 3.1h (Insert statement)***

Insert two rows of data into the table. Notice the data that is being inserted into the table is case-sensitive but the syntax is not. Will insert values into top table

INSERT INTO patient VALUES (11,'John', 'Smith', 'm','01-FEB-1970', 55000);

­­­­­­­­­­­­­­

***Example 3.1k (Delete)***

The table will still exist but its contents will be deleted. The data can still be recovered if an implicit or explicit commit has not been implemented. Delete can be applied to all or specific rows in a table. Notice no asterisk

DELETE FROM patient;

***Example 3.1l (Truncate)***

The table will still exist but its contents will be deleted. The data cannot be recovered. It is a lot faster than delete. Unlike delete, it is applied to all the rows in the table.

TRUNCATE TABLE patient;

What is wrong?

Delete \* FROM patient;

It has \*

**3.2 Adding columns using ALTER command**

Used for structural changes to a table. Example, you might need to add a column, delete, or change column’s size. Changes are made with ALTER TABLE.

ALTER TABLE tablename ADD|MODIFY|DROP| columnname (definition);

Using an ADD clause with the ALTER TABLE command allows a user to add a new column to a table.

***Example 3.2a (Additional column)***

--This statement adds marital\_status as a column to the table patient. The datatype of this new

--column is CHAR.

ALTER TABLE patient ADD marital\_status CHAR;

***Example 3.2b (Adding multiple columns)***

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Adding multiple columns at the  same type --For multiple columns, the syntax looks like the create table statement.  ALTER TABLE patient ADD  (  Height NUMBER,  Weight NUMBER  );  3.3 Modifying using ALTER  command  To change an existing column’s  definition, we use MODIFY.  With the ALTER TABLE:   * Changing the column size ( increase or   decrease)   * Changing the datatype ( such as * VARCHAR2 to CHAR) * Changing or adding the default value   of a column ( such as DEFAULT  SYSDATE)  Be aware of three rules when modifying existing columns:   * A column must be as wide as the   data fields it already contains.   * If a NUMBER column already   contains data, you can’t decrease  the column’s precision or scale.   * Changing the default value of a   column doesn’t change the  values of data already in the table  ***Example 3.3a (Modify)***  As long as the table is empty, the type can  be modified and the constraint can be  changed without any problems.    ALTER TABLE patient MODIFY fname  VARCHAR2(19);  ***Example 3.4b (Dropping columns)***  Getting rid of multiple columns.  ALTER TABLE patient DROP (annual\_salary,  marital\_status);  ***Example 3.4c (Set unused)***   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Setting columns to unused and then dropping  them (Cannot be rolled back). --Cannot recover  the data  ALTER TABLE patient SET UNUSED (DOB, gender);  DESC patient;  --Cannot recover the data  ALTER TABLE patient DROP UNUSED  COLUMNS;  Renaming column using ALTER command  ***Example 3.5a (Renaming a column)***   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Renames a column from fname to first\_name.  Only one column can be renamed at a time.  ALTER TABLE patient RENAME  COLUMN fname TO first\_name;    3.6 Constraints  constraint in two ways: at the  column level or the table level.  constraint at the column level means the constraint’s definition is included as part of the column definition,  similar to assigning a default value  to a column  Creating a constraint at the table  level means the constraint’s  definition is separate from the  column definition.  The main difference in the  syntax of a column- level constraint  and a table- level constraint is that you  provide column names for the table- level constraint  at the end of the constraint definition inside  parentheses, instead of at the beginning  of the constraint definition.  ***3.7 Primary key constraint***  Example 3.7a (Primary key at the column level)  The constraint makes certain the columns  identified as the table’s primary key are  unique and do not contain NULL values.  DROP TABLE patient;  CREATE TABLE Patient  (  Patient\_id NUMBER PRIMARY KEY,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  );  INSERT INTO patient VALUES (11,'John', 'Smith');  ***Example 3.7b (User\_constraints table)***   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Examining the system table user\_constraints --Constraint\_type (p)  stands for primary key. The constraint name is generated by the  system.  --In the SQL statement, PATIENT must be typed as upper-case because that is how it is stored in  --ORACLE system table.  SELECT table\_name, constraint\_name,  constraint\_type FROM  user\_constraints WHERE table\_name='PATIENT';  ***Example 3.7d (Primary key at the table level)***   |  |  | | --- | --- | | Creating a primary key constraint at the table level. DROP TABLE  patient;  --Use the CONSTRAINT keyword when giving a name to a constraint.  --The constraint is created at the table level after all column  definitions.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20),  CONSTRAINT patient\_patient\_id\_pk PRIMARY KEY (patient\_id)  );  DROP TABLE patient;  ***Example 3.7e (Composite primary key)***   |  | | --- | | This example creates a composite primary key, with the  assumption that the combination of fname and lname  are unique. Composite key cannot be created at the column level. DROP TABLE patient;  --Invalid example: Cannot have two primary keys.  --The syntax for composite primary key requires a table level  syntax.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20) PRIMARY KEY,  Lname VARCHAR2(20) PRIMARY KEY  );  --Here is the syntax for a composite primary key with a constraint  name.  --Notice the constraint keyword is used to give a name to the constraint.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20),  CONSTRAINT patient\_fname\_lname\_pk  PRIMARY KEY (Fname, Lname)  ); | |   ***Example 3.7f (Using alter table command)***   |  |  | | --- | --- | | Creating a primary key using the alter command DROP TABLE  patient;  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  );  --Can create a primary key using alter table statement.  ALTER TABLE patient ADD PRIMARY KEY  (patient\_id);  3.8 Unique constraint  ***Example 3.8a (Unique key at the column level)***   |  | | --- | | Creating a unique key constraint at the  column level DROP TABLE patient;  --Use the UNIQUE keyword to create a  candidate or unique key.  --A table can have many unique keys. Also,  unlike a primary key which cannot-be NULL,  a unique key  --column can have NULLs.  CREATE TABLE Patient  (  Patient\_id NUMBER UNIQUE,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  ); | | |   ***Example 3.8d (Unique key at the table level)***   |  | | --- | | Creating a unique key constraint at the table level DROP TABLE patient;  --Unique key constraint can also be created at the table level just like a primary key.  --The constraint keyword is used to give a name to the unique key.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20),  CONSTRAINT patient\_patient\_id\_uk UNIQUE (patient\_id)  );  ***Example 3.8e (Composite unique key)***  This example creates a composite unique key  with the assumption that the combination of  fname and lname must be unique. Composite key cannot be created at the column level. |   DROP TABLE patient;  --Unlike a primary key, a table can have multiple unique keys. This is not a composite, unique key. Fname and  --lname are unique by themselves. The system generates a name.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20) UNIQUE,  Lname VARCHAR2(20) UNIQUE  );  --Composite unique key has the same basic syntax as a composite primary key.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20),  CONSTRAINT patient\_fname\_lname\_uk UNIQUE (Fname, Lname)  );  ***Example 3.8f (Creating unique with alter table***  ***command)***  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  );  ALTER TABLE patient ADD CONSTRAINT  patient\_patient\_id\_uk UNIQUE (  patient\_id);  --Can use MODIFY for a unqique constraint without  the constraint keyword.  ALTER TABLE patient MODIFY  patient\_id UNIQUE;  --Use the constraint keyword to give a name to the  constraint.  ALTER TABLE patient MODIFY  patient\_id CONSTRAINT  patient\_patient\_id\_uk UNIQUE;    ***Example 3.8g (Using alter to create composite***  ***unique key)***   |  | | --- | | This example creates a composite unique key using the alter command. DROP TABLE patient;  CREATE TABLE Patient  (  Patient\_id NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  );  --Creates a composite unique key after the table  has been created.  ALTER TABLE patient ADD UNIQUE  (fname, lname); |   ***Example 3.9a (Column v. table level check***  ***constraint)***   |  | | --- | | Column level versus table level without a  constraint name DROP TABLE patient;  --Use the Check keyword to create a check  constraint. The name for this constraint will be  --generated by the system. This constraint is  created at the column level because the comma  --appears after the column name, datatype  and the actual check constraint.  CREATE TABLE Patient  (  Patient\_id NUMBER,  Height NUMBER CHECK (height>10),  Fname VARCHAR2(20),  Lname VARCHAR2(20)  ); |   ***Example 3.9d (Using alter table to create check***  ***constraint)***   |  | | --- | | Creating a check constraint using the alter  statement with and without a constraint name  DROP TABLE patient;  CREATE TABLE Patient  (  Patient\_id NUMBER,  Height NUMBER,  Fname VARCHAR2(20),  Lname VARCHAR2(20)  );  ALTER TABLE patient MODIFY  height CONSTRAINT  patient\_height\_ck CHECK(height>10); | | | |
| Entity and Attribute  Entity   * An object with a physical existence or conceptual existence * e.g. a person, a company * Notation   Attributes   * Properties that describe entities * e.g. Name of an employee * Notation   Attribute  -simple vd. Composite attribute   * Simple (atomic) attribute   + Attribute that are not divisible   + e.g. SSN, ZIP code   + Notation * Composite Attribute   + can be divided into smaller subparts   + e.g. Address   + Notation   Attribute  -single value vs Multivalued attribute   * Single-valued attribute   + e.g. Age of a person * Multivalued attribute   + e.g. College degree   + Notation   Attribute  -Single value vs. Multivalued attribute   * Single-valued attribute   + e.g. Age of a person * Multivalued attribute   + e.g. College degree   + Notation |

**A screenshot of a cell phone

Description automatically generated**

**HW#5**

**1A Create a student table that will hold the following data. Make sure you assign a primary key**

CREATE TABLE student

(

Ssn VARCHAR2(20) PRIMARY KEY,

Lname VARCHAR2(20),

Fname VARCHAR2(20),

Age NUMBER,

Salary NUMBER,

Dob NUMBER

);

**1B) After the table has been created add a candidate key based on lname and fname.**

Note: Candidate and unique key are the same thing

ALTER TABLE student ADD CONSTRAINT stident\_lname\_fname UNIQUE (Lname,Fname);

**1C) After the table has been created add a check constraint such that the age is greater than 10 but less than 50. Provide a name for the check constraint.**

ALTER TABLE student MODIFY age CHECK (AGE>=10 AND AGE<50);

**1D) After the table has been created add a column called address.**

ALTER TABLE STUDENT ADD Address varchar2(30);

**1E) After the table has been created, modify the dob column to be of datatype date and also not null**

ALTER TABLE student MODIFY Dob DATE NOT NULL;

**1F) Create a composite index on ssn and dob**

CREATE INDEX ssn\_dob\_idx ON Student (ssn,dob)

**1G) After the table has been created add a column called transferable with a not null constraint. Do not assign a name to the constraint**

ALTER TABLE student ADD Transeferable VARCHAR2(20) NOT NULL;

**1H) After the table has been created add a check constraint on the column transferable to allow only ‘y’,’Y’,’n’,’N’. Give the constraint a name.**

ALTER TABLE student ADD CONSTRAINT col\_trans CHECK(Transeferable='y' AND Transeferable='Y' AND Transeferable='n' AND Transeferable='N');

ALTER TABLE student ADD CONSTRAINT col\_trans CHECK(transeferable IN(‘Y’,’y’,’N’,’n’));

**2A) Create a second table called class that will hold the following data. You decide what the data types are going to be.**

CREATE TABLE class (

code NUMBER,

descrip varchar2(20) UNIQUE NOT NULL

);

CREATE TABLE class

(

code NUMBER,

descrip varchar2(20) NOT NULL,

UNIQUE (descript)

);

**2B) After the table has been created add the primary key. Give the constraint a name**

ALTER TABLE class MODIFY Code Primary Key;

**2C) Create an index on class description**

CREATE INDEX class\_descp\_idx ON class (descrip);

**3A) Create a third table called student\_class. This table is an association table that contains information on the different class that the students are taking. You figure out what the columns should be. It should contain only two columns.**

CREATE TABLE student\_class

(

ssn varchar2(20),

classCode NUMBER

)

**3B After the table has been created add the primary key constraint (Name the constraint)**

ALTER TABLE student\_class ADD CONSTRAINT student\_class\_pk PRIMARY KEY(ssn, classCode);

**3C After the table has been created add the foreign key constraint(s) (Name the constraint(s))**