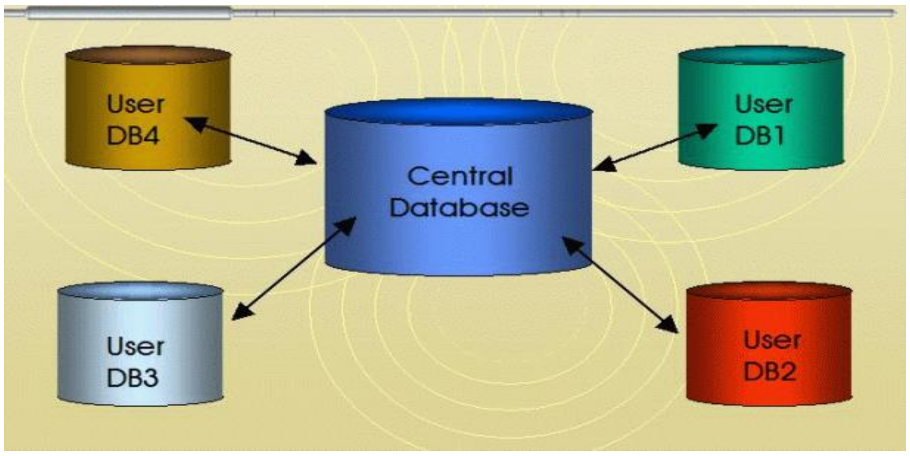
Chapter 1(Getting started)



"A compromise is an agreement whereby both parties get what neither of them wanted"

Objectives for this book

- Understand relational database fundamentals
- Create databases
- Understand the normalization process
- Recognize poor table design
- Understand the principles of adding, deleting, updating, and sorting records within a table
- Create queries
- Import/Export data

1.1 Database

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

- Hierarchical model
- Network model
- Relational model
- Object Relational model
- Object Oriented Relational model.

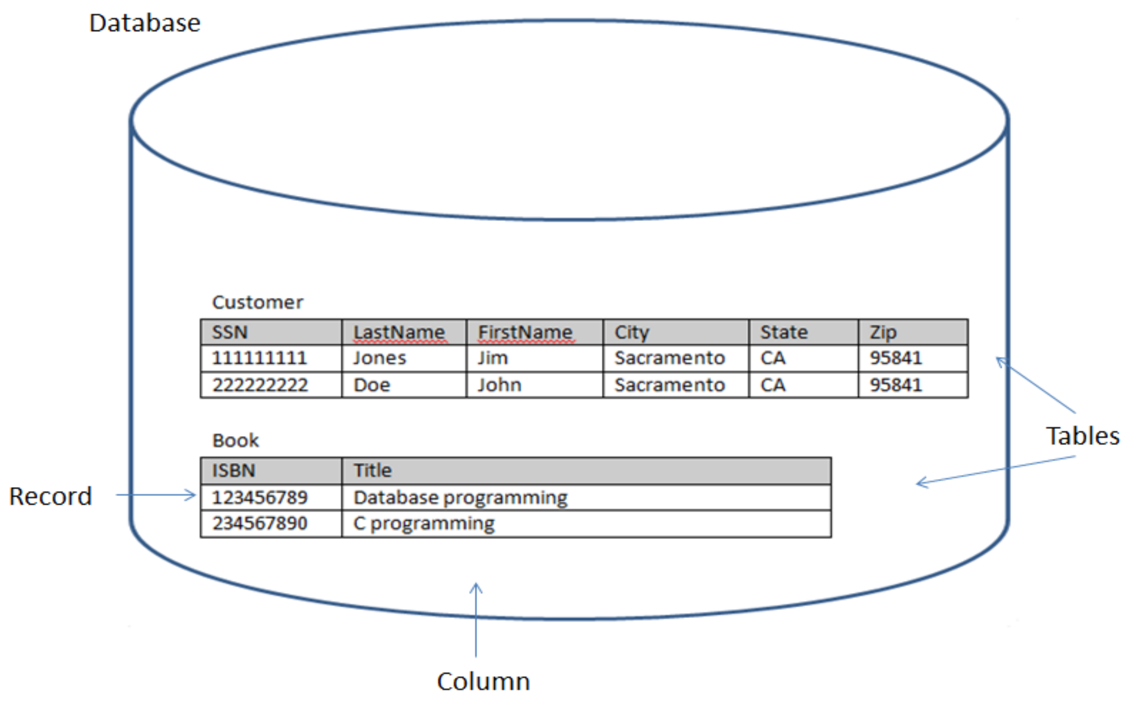
The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. The Network model allows for more than one parent to be associated with a child.An object database (also object-oriented database) is a database model in which information is represented in the form of objects as used in object-oriented programming. An object-relational database (ORD) is similar to a relational database, but with an object-oriented database model and can be said to provide a middle ground between relational databases and object-oriented databases. In object-relational databases, the approach is essentially that of relational databases. Our focus is on the relational model.

1.2 Relational model

A relational database matches data by using common characteristics found within the data set. The resulting groups of data are organized and are much easier for many people to understand.

- **Data hierarchy**: ordering of data types by size
  - **Field**: group of characters forming a single data item
    - "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 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.



Customer

SSN	LastName	FirstName	City	State	Zip
111111111	Jones	Jim	Sacramento	CA	95841
222222222	Doe	John	Sacramento	CA	95841

Book

ISBN	Title
123456789	Database programming
234567890	C programming

1.3 DBMS

"That is what learning is. You suddenly understand something you've understood all your life, but in a new way."

A Database Management System (DBMS) is a set of computer programs that controls the creation, maintenance, and the use of a database.

- **Database management software**
    - Create table descriptions
    - Identify keys
    - Add, delete, and update records within a table
    - Sort records by different fields
    - Write questions to select specific records for viewing
    - Write questions to combine information from multiple tables
    - Create reports
    - Secure the data
        - Providing data integrity
        - Recovering lost data
        - Avoiding concurrent update problems
            - Two users make changes to the same record
            - Lock: mechanism to prevent changes to a record for some period of time
        - Providing authentication and permissions
            - Storing and verifying passwords
            - Using biometric data to identify users
            - settings that determine what actions a user is allowed to perform
        - Encryption (For data security)
            - Prevents use of the data by unauthorized users

1.4 RDBMS

The software used to do this grouping is called a relational database management system (RDBMS).

1.5 DBAvs. DA

"A bad beginning makes a bad ending"

A database administrator (DBA) is a person responsible for the implementation, maintenance and repair of an organization's database.

A data analyst is a person responsible for analyzing data requirements within an organization and modeling the data.

1.6 Some database vendors

- IBM DB2
- Microsoft SQL Server
- Informix
- Sybase Inc.
- Oracle
  - Based in Redwood, California
  - Leader in the worldwide relational and object-relational database management systems software market.

Our focus will be on the Oracle product.

1.7 SQL

It is referred to as Structured Query Language. It is a database computer language designed for managing data in relational database management systems (RDBMS). This language was originally based upon relational algebra and calculus. It is comprised of several sub-languages:

Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

Data Manipulation Language (DML) statements are used for managing data within schema objects. Some examples:

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

Data Control Language (DCL) statements. Some examples:

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

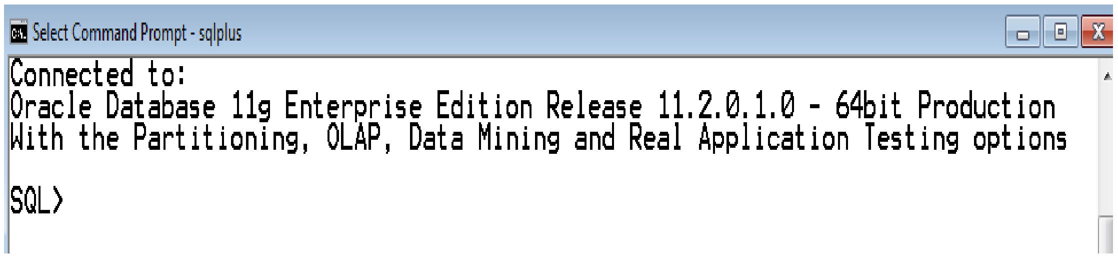
- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use

❑ Example of a SQL statement

SELECT * FROM customer WHERE city='Paris';

1.8 SQL*Plus

SQL*Plus is an Oracle command-line utility program that can run SQL and PL/SQL commands interactively or from a script.



```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

1.9 SQL vs. SQL*Plus

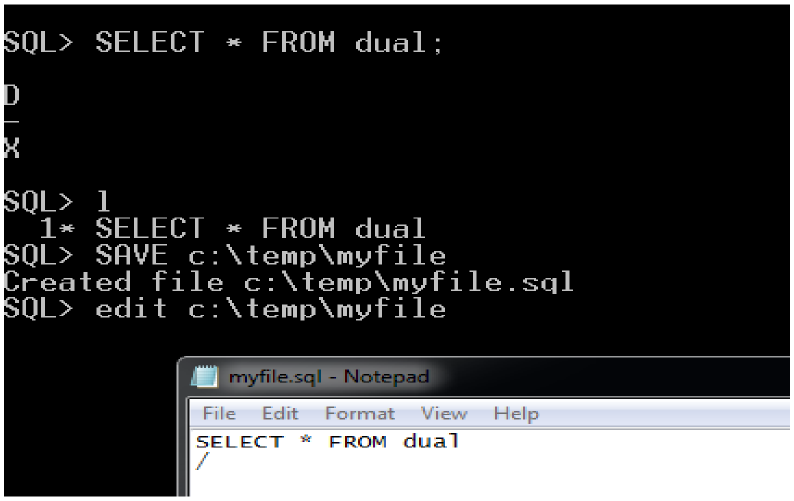
SQL commands end with a semi-colon. They work directly with the database. SQLPlus commands do not need a semi-colon. SQLPlus is a text based enviornment that is used to type in SQL commands It allows you to interact with the database.

1.10 Some SQL*Plus commands

All these commands are oracle specific. They are not SQL commands. These commands are not case sensitive. Unlike SQL commands, SQLPlus commands do not end with semi-colon. The SQLPlus buffer will only store the last SQL command. The SQLPlus buffer does not store any SQLPlus commands.

/	Runs the SQL statement in the buffer.
$	Execute operating system command from within SQLPlus Example: $ del c:\temp\test.SQL
/* */	These are multi-line comment symbols. Example : /* some comments This is my name. */
--	This is also a comment but only for one line. Example: -- This is a comment
APPEND	Appends to the end of your SQL buffer. Example: append user_tables Appends user_tables to the end of the same line.
CHANGE /from /to	Correct any spelling errors. **Note**: List the line number you want to change first and then issue the change command. Example: change /stff /stuff or c/stff/stuff Will change stff to stuff.
clear buffer	Clears the buffer and lets you start over.
CLEAR SCREEN	Clears the screen.

CONNECT	Logs you in as a different user. Example: connect po7 Logs you in as po7 and then prompts you for a password.
DEL line number	Will delete the line. Example: del 3 Deletes line 3 from the buffer.
DISCONNECT	Ends oracle session.
EDIT filename	Launches notepad with either a new file or an existing file. Example: edit c:\temp\test Note: The file extension will automatically be .SQL.
HOST	Execute operating system command from within SQLPlus. Example: host del c:\temp\test.sql NOTE: Have to identify both the file name and the file extension because you are issuing a command from the operating system which does not know what extension you have.
INPUT	Will add whatever text you want to the next line in the buffer. Example: input user_tables Will add the word user_tables to the next line in your buffer.
LIST or **L**	Will list everything in the SQLPlus buffer. Example: list or l
LIST line number	Will list the statement associated with the line number. Example: list 2 or l2 Lists only statement in line 2.
REM	Remark or comment. Example: REM This is a comment
Run or R	Runs whatever is in the buffer.
SAVE Filename	Saves the contents of the buffer into a new file. Example: save c:\temp\myfile Note: If file already exists then save with replace option Example: save c:\temp\test replace

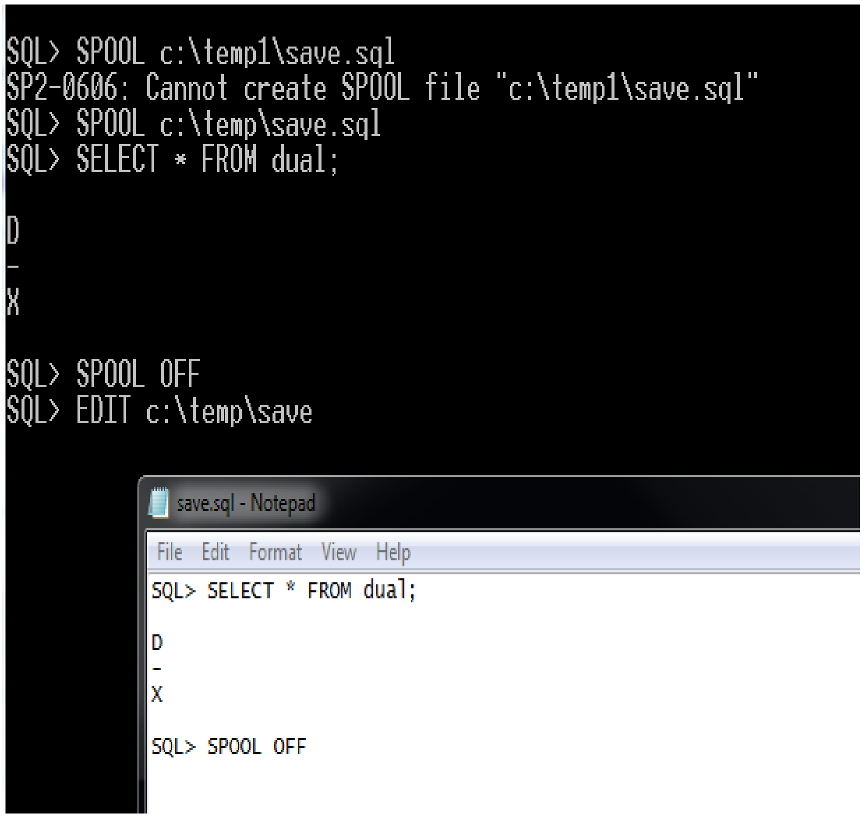


```
SQL> SELECT * FROM dual;

D
-
X

SQL> l
  1* SELECT * FROM dual
SQL> SAVE c:\temp\myfile
Created file c:\temp\myfile.sql
SQL> edit c:\temp\myfile
```

```
SELECT * FROM dual
/
```

SPOOL filename Will save everything that you see on the SQLPlus screen to a file.
Example: spool c:\temp\save.sql
Writes to a file called c:\temp\save.sql. Make sure the directory exists and that you have permission to write to directory, otherwise you will get an error message as shown.

SPOOL OFF Stops writing to the file.



```
SQL> SPOOL c:\temp1\save.sql
SP2-0606: Cannot create SPOOL file "c:\temp1\save.sql"
SQL> SPOOL c:\temp\save.sql
SQL> SELECT * FROM dual;

D
-
X

SQL> SPOOL OFF
SQL> EDIT c:\temp\save
```

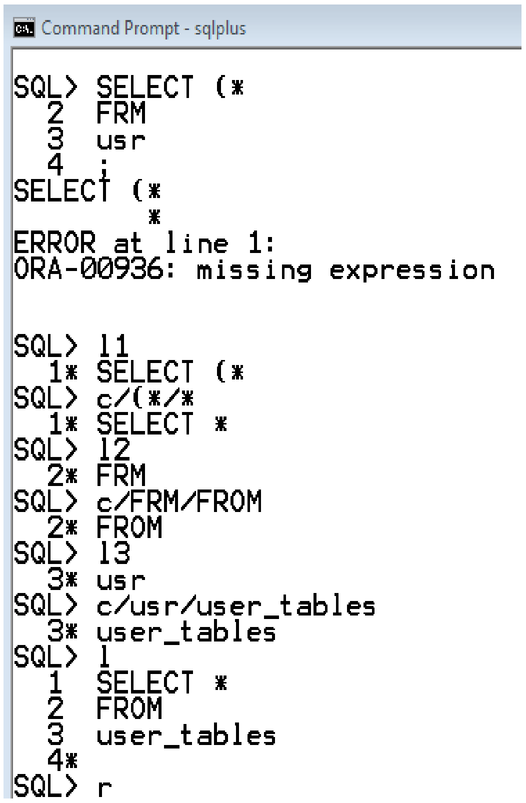
```
SQL> SELECT * FROM dual;

D
-
X

SQL> SPOOL OFF
```

START filename	Runs the contents of the file. Example:Start c:\temp\test Note: If the file does not contain a semi-colon or / then you will have to type in / to run the contents of the file.

<u>CAUTION: SPOOL vs (SAVE or EDIT)</u>

When you edit or save a file, only the contents of the buffer, which contains SQL statements, will be in the file. However, the spool command will save everything that you see on your output screen to a file regardless of whether it is SQL or not.

Example 1.10a



```
SQL> SELECT (*
  2  FRM
  3  usr
  4  ;
SELECT (*
        *
ERROR at line 1:
ORA-00936: missing expression


SQL> l1
  1* SELECT (*
SQL> c/(*/*
  1* SELECT *
SQL> l2
  2* FRM
SQL> c/FRM/FROM
  2* FROM
SQL> l3
  3* usr
SQL> c/usr/user_tables
  3* user_tables
SQL> l
  1  SELECT *
  2  FROM
  3  user_tables
  4*
SQL> r
```

1.11 Additional SQL*Plus commands

SHOW ALL	Shows the status of all SQLPlus variables.
SHOW ECHO	Shows the status of the variable echo. Example: SET ECHO OFF Example: SET ECHO ON *Use the set command to assign a new value to a variable.*
SHOW LINESIZE	Shows the status of linesize which is the width of screen. Example: SET LINESIZE 50
SHOW PAGESIZE	Shows the status of pagesize which is the length of screen.
SET PAGESIZE 20	Sets the number of lines that make up a page. *The header and the footer appear on top and bottom of the report, respectively. The number of lines per page determine when the header, footer and also the column headings appear in a report.*
COLUMN	Shows the status of all columns. All changes to columns will last for the oracle session.
CLEAR COLUMNS	Reset all column formatting to default.

COLUMN original_column_name HEADING new_column_name

Modify the column name.

COLUMN original_column_name FORMAT A5

Format text columns to display five characters.

COLUMN salary FORMAT99999.99

Format numeric columns. Make sure you consider the largest number in your table.

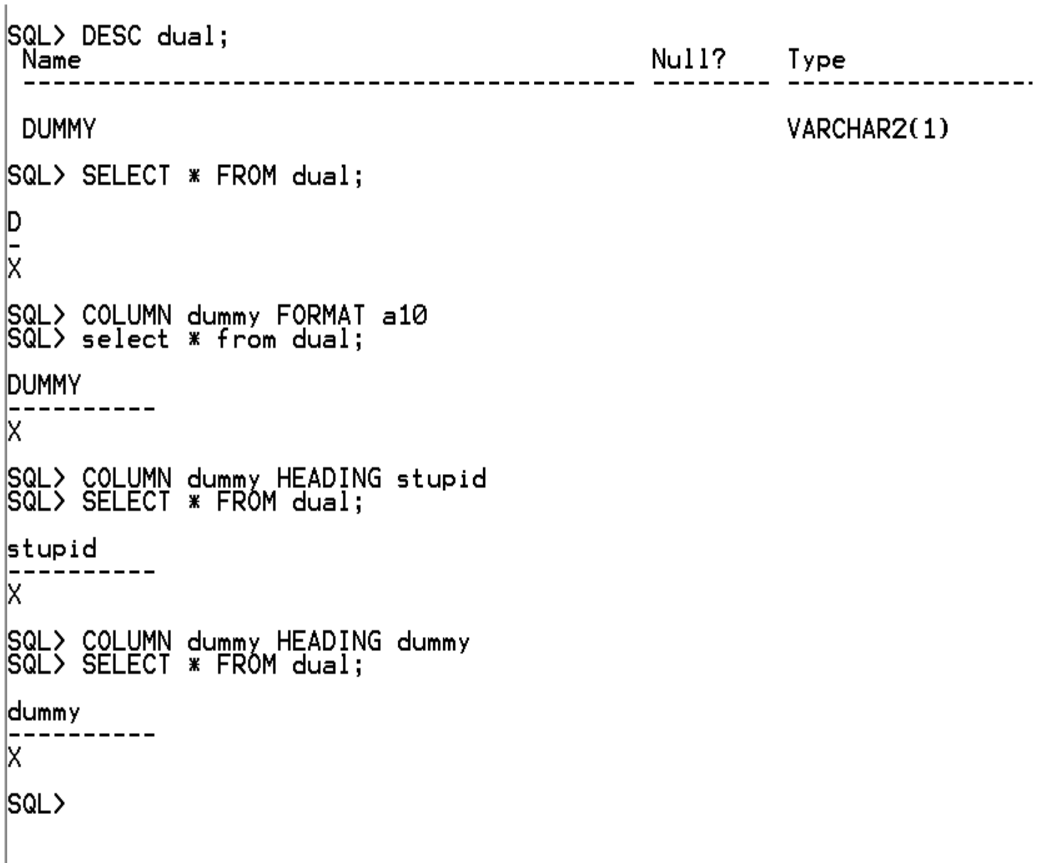
COLUMN original_column_name TRUNCATED

Truncate if column size is beyond the width you set with the format command.

COLUMN original_column_name WORD_WRAPPED

Word_wrapped is the default

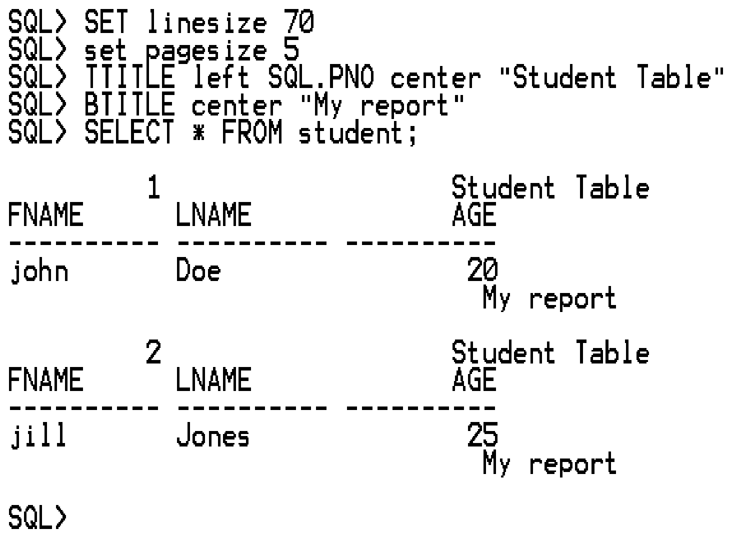
Example 1.11a



```
SQL> DESC dual;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 DUMMY                                              VARCHAR2(1)

SQL> SELECT * FROM dual;

D
-
X

SQL> COLUMN dummy FORMAT a10
SQL> select * from dual;

DUMMY
----------
X

SQL> COLUMN dummy HEADING stupid
SQL> SELECT * FROM dual;

stupid
----------
X

SQL> COLUMN dummy HEADING dummy
SQL> SELECT * FROM dual;

dummy
----------
X

SQL>
```

FEEDBACK	Reports on the number of rows retrieved.
SHOW FEEDBACK	**S**how the status of feedback variable.
SET FEEDBACK 2	Show number of rows if it is 2 or greater.
TTITLE	Header
TTITLE CENTER “my report”	Puts title on top of every page.
BTITLE	Footer
BTITLE LEFT SQL.PNO “my report”	Puts footer on the bottom of every page. The footer is left justified. Displays the page number using sql.pno

Example 1.11b



```
SQL> SET linesize 70
SQL> set pagesize 5
SQL> TTITLE left SQL.PNO center "Student Table"
SQL> BTITLE center "My report"
SQL> SELECT * FROM student;

         1                   Student Table
FNAME      LNAME                    AGE
---------- ---------- ----------
john       Doe                       20
                               My report

         2                   Student Table
FNAME      LNAME                    AGE
---------- ---------- ----------
jill       Jones                     25
                               My report

SQL>
```

1.12 PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle's Procedural extension language. PL/SQL's general syntax resembles that of Ada.

✓ *CHECK 1A*

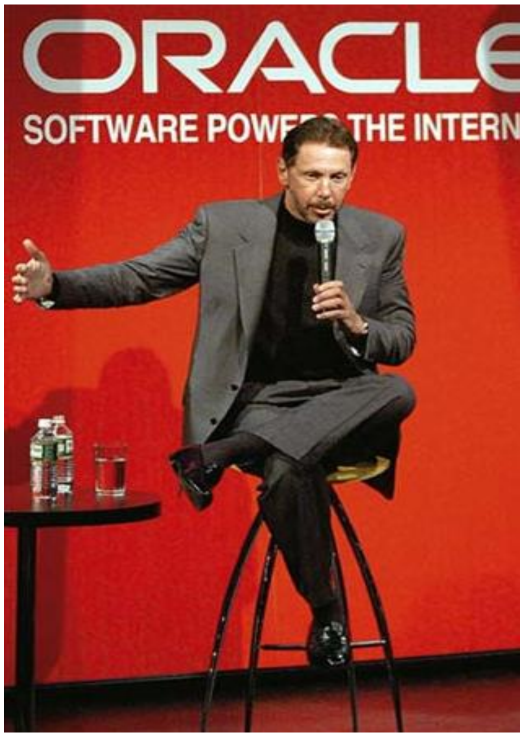
1. What is the difference between SQL*Plus, SQL and PL/SQL?
2. What is the SQL*Plus buffer?
3. What are the SQL sub-languages?
4. What are some examples of SQL*Plus variables?
5. What are some examples of SQL*Plus commands?
6. How do you delete a file from within SQL*Plus?

"*A man of words and not of deeds is like a garden full of weeds.*"

SummaryExamples

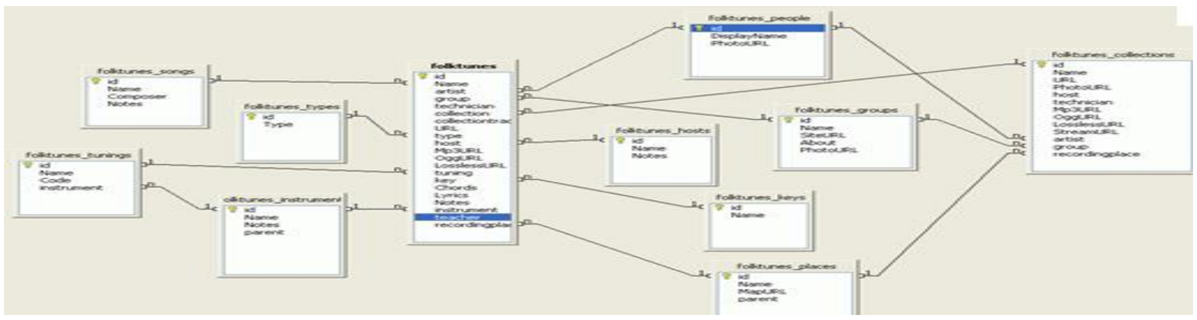
```
-- or REM  or /*   */ can be used as comment symbols.
-- An erroneous SQL statement that will be corrected using SQLPlus commands.
SLECT * FROM
user
table;

-- List contents of the buffer, which is the very last SQL statement (not SQLPlus).
L or LIST


-- Point to the first line.
L1

-- Change from SLECT to SELECT (Like find and replace).
C/SLECT/SELECT   or CHANGE/slect/select

--Get rid of the second line. All other lines are renumbered.
DEL 2


--Run contents of the buffer
RUN, R,  or /

--Save the contents of buffer to a file called hi.sql.
SAVE c:\temp\hi


--Edit the contents of the saved file.
EDIT c:\temp\hi

--Execute the contents of a file.
START c:\temp\hi


--Clear screen.
CLEAR SCREEN

--Clear the buffer.
CLEAR BUFFER

-- $ or host gets access to operating system commands.
$ del c:\temp\hi.sql   or HOST del c:\temp\hi.sql

--Show what the SQLPlus variable, pagesize, is set to.
SHOW PAGESIZE

--Set pagesize to 10
SET PAGESIZE 10


--Set linesize to 100
SET LINESIZE 100
```

```
--Set top title to (hello). Center it and include the page number.
TTITLE CENTER "hello" SQL.PNO

--Set footer to (goodbye) and include the page number.
BTITLE LEFT "goodbye" SQL.PNO

--Change the size of the column to 40 characters.
COLUMN table_name A40

--Display original column name, lname to lastname for the Oracle session.
COLUMN  lname HEADING lastname

--Direct contents of the SQLPLus screen to a file.
SPOOL c:\temp\redirect.txt

--Turn off spooling.
SPOOL OFF

--Disconnect from database.
DISCONNECT

--Connect as a different user.
CONNECT username/password   or connect username
```



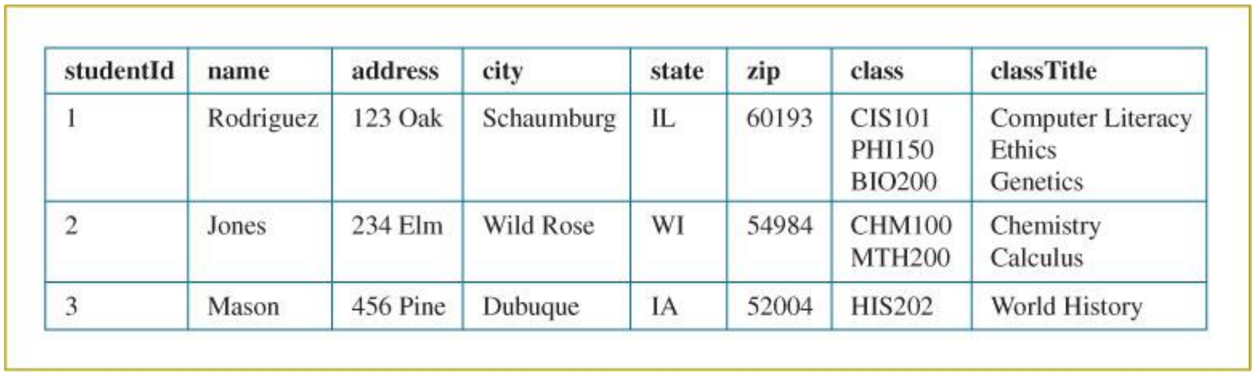
In 1990, Oracle laid off 10% (about 400 people) of its work force because it was losing money. This crisis, which almost resulted in Oracle's bankruptcy, came about because of Oracle's "up-front" marketing strategy, in which sales people urged potential customers to buy the largest possible amount of software all at once. The sales people then booked the value of future license sales in the current quarter, thereby increasing their bonuses. This became a problem when the future sales subsequently failed to materialize. Oracle eventually had to restate its earnings twice, and also to settle out of court class action lawsuits arising from its having overstated its earnings.

Chapter 2 (Database Design)



"Middle age is when you are warned to slow down by a doctor instead of a policeman."

2.1 Poor Table Design



studentId	name	address	city	state	zip	class	classTitle
1	Rodriguez	123 Oak	Schaumburg	IL	60193	CIS101 PHI150 BIO200	Computer Literacy Ethics Genetics
2	Jones	234 Elm	Wild Rose	WI	54984	CHM100 MTH200	Chemistry Calculus
3	Mason	456 Pine	Dubuque	IA	52004	HIS202	World History

2.2 Normalization

The process of refining tables, keys, columns, and relationships to create an efficient database is called *normalization*. 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.

Most designs implement up to the third normal form.

2.4 Primary key, Foreign key, Candidate or Alternate key

"A drowning man will catch at a straw"

Three attributes are the heart of data normalization:

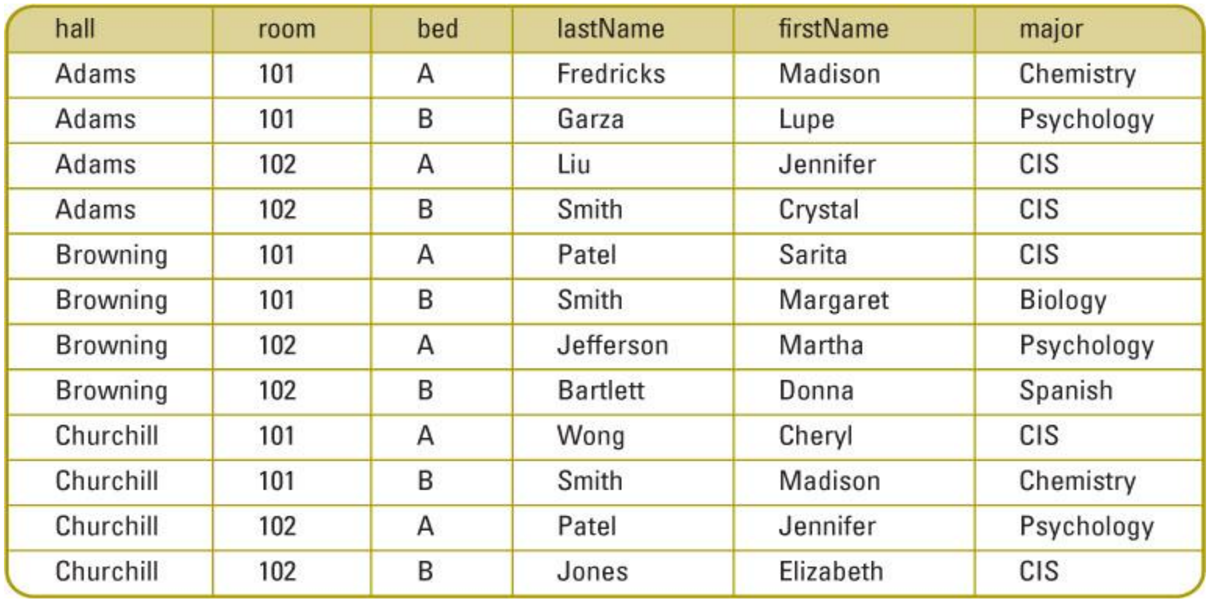
- Primary key
  - A field whose values are unique for each record in a table
  - The Primary Key ensures that no two records in a database contain the same value for that field
  - Creating relationships between tables
  - May be composed of one or multiple columns
    - **Called a Compound or a composite 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.

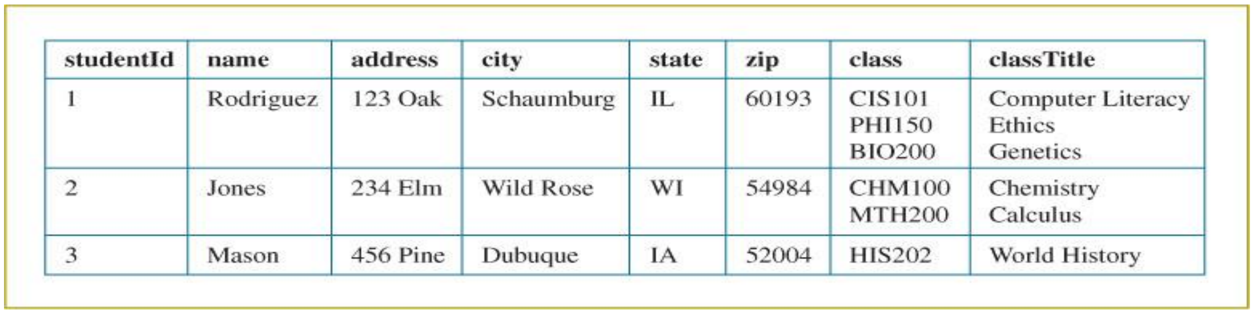
Identify primary key:



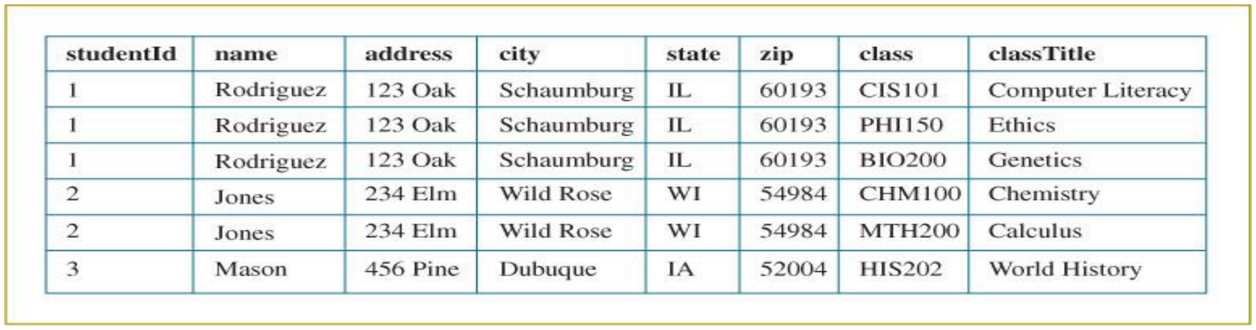
hall	room	bed	lastName	firstName	major
Adams	101	A	Fredricks	Madison	Chemistry
Adams	101	B	Garza	Lupe	Psychology
Adams	102	A	Liu	Jennifer	CIS
Adams	102	B	Smith	Crystal	CIS
Browning	101	A	Patel	Sarita	CIS
Browning	101	B	Smith	Margaret	Biology
Browning	102	A	Jefferson	Martha	Psychology
Browning	102	B	Bartlett	Donna	Spanish
Churchill	101	A	Wong	Cheryl	CIS
Churchill	101	B	Smith	Madison	Chemistry
Churchill	102	A	Patel	Jennifer	Psychology
Churchill	102	B	Jones	Elizabeth	CIS

2.5 First Normal Form (1NF)

- Unnormalized: table contains repeating groups
- Repeating group: subset of rows in a table all depend on the same key
- Table in 1NF contains no repeating groups of data
- Primary key attributes are defined
- Atomic attributes: columns contain undividable pieces of data
- In 1NF, all values for intersecting row and column must be atomic



studentId	name	address	city	state	zip	class	classTitle
1	Rodriguez	123 Oak	Schaumburg	IL	60193	CIS101 PHI150 BIO200	Computer Literacy Ethics Genetics
2	Jones	234 Elm	Wild Rose	WI	54984	CHM100 MTH200	Chemistry Calculus
3	Mason	456 Pine	Dubuque	IA	52004	HIS202	World History



studentId	name	address	city	state	zip	class	classTitle
1	Rodriguez	123 Oak	Schaumburg	IL	60193	CIS101	Computer Literacy
1	Rodriguez	123 Oak	Schaumburg	IL	60193	PHI150	Ethics
1	Rodriguez	123 Oak	Schaumburg	IL	60193	BIO200	Genetics
2	Jones	234 Elm	Wild Rose	WI	54984	CHM100	Chemistry
2	Jones	234 Elm	Wild Rose	WI	54984	MTH200	Calculus
3	Mason	456 Pine	Dubuque	IA	52004	HIS202	World History

2.6 Second Normal Form (2NF)

- Partial key dependencies: column depends on only part of the key
- For 2NF:
  - Database must already be in 1NF
  - All non-key fields must be dependent on the entire primary key
- Eliminate partial key dependencies by creating multiple tables
- Improvements over 1NF:
  - Eliminate update anomalies
  - Eliminate redundancies
  - Eliminate insert anomalies
  - Eliminate delete anomalies

2.7 Third Normal Form (3NF)

- All redundancies and anomalies are removed
- Normalization summary:
  - 1NF: no repeating groups
  - 2NF: 1NF plus no partial key dependencies
  - 3NF: 2NF plus no transitive dependencies

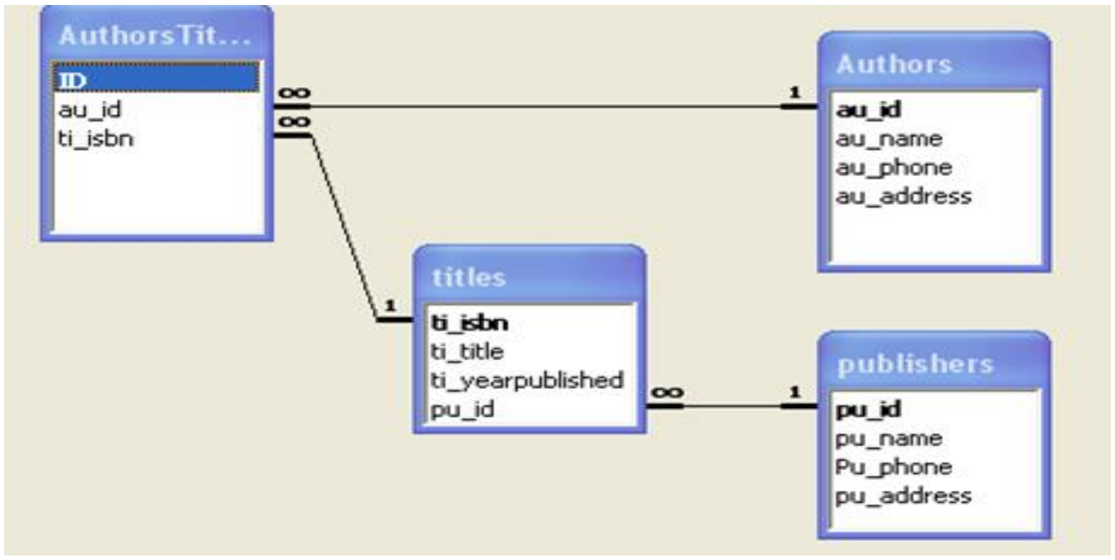
2.8 Entity Relationship Diagrams (ER)

Entity Relationship (ER) is a conceptual model that shows the structural organization of entities, relationships, and attributes. Three types of relationships:

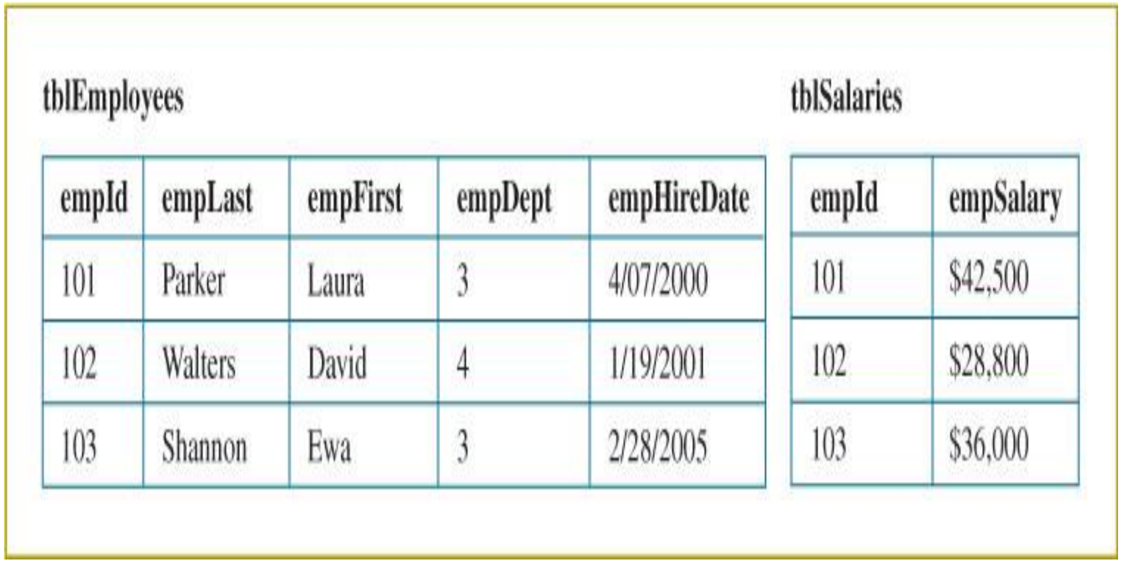
- ■ One-to-one: one instance of an entity (A) is associated with one other instance of another entity (B).
  - Row in one table corresponds to exactly one row in another table
  - Least frequently encountered relationship
  - Tables could be combined into a single table
  - Keep tables separate for security purposes

- ■ One-to-many: one instance of an entity (A) is associated with zero, one or many instances of another entity (B), but for one instance of entity B there is only one instance of entity A.
  - Most common type of table relationship
  - Row in one table related to one or more rows in another table
  - “One” side is the base table, “Many” side is the related table
  - Primary key is used for the join
  - Foreign key: Field in one table which is primary key in another table

- Many-to-many: one instance of an entity (A) is associated with one, zero or many instances of another entity (B), and verse versa.
  - Multiple rows in each table can correspond to multiple rows in the other table.
  - Many to many relationships should be eliminated through the addition of a bridge or an association table which results in one to many relationships.



2.9 One to One



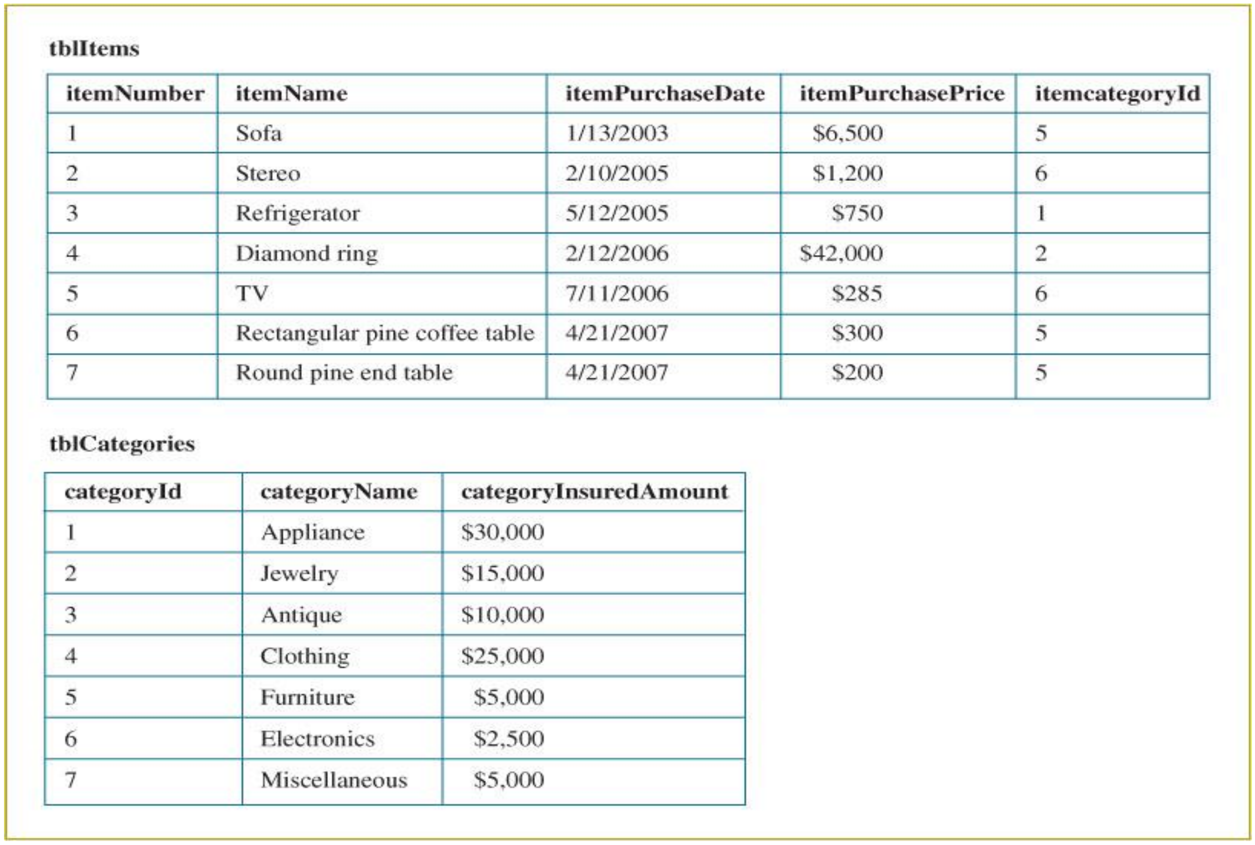
tblEmployees

empId	empLast	empFirst	empDept	empHireDate
101	Parker	Laura	3	4/07/2000
102	Walters	David	4	1/19/2001
103	Shannon	Ewa	3	2/28/2005

tblSalaries

empId	empSalary
101	$42,500
102	$28,800
103	$36,000

2.10 One to Many

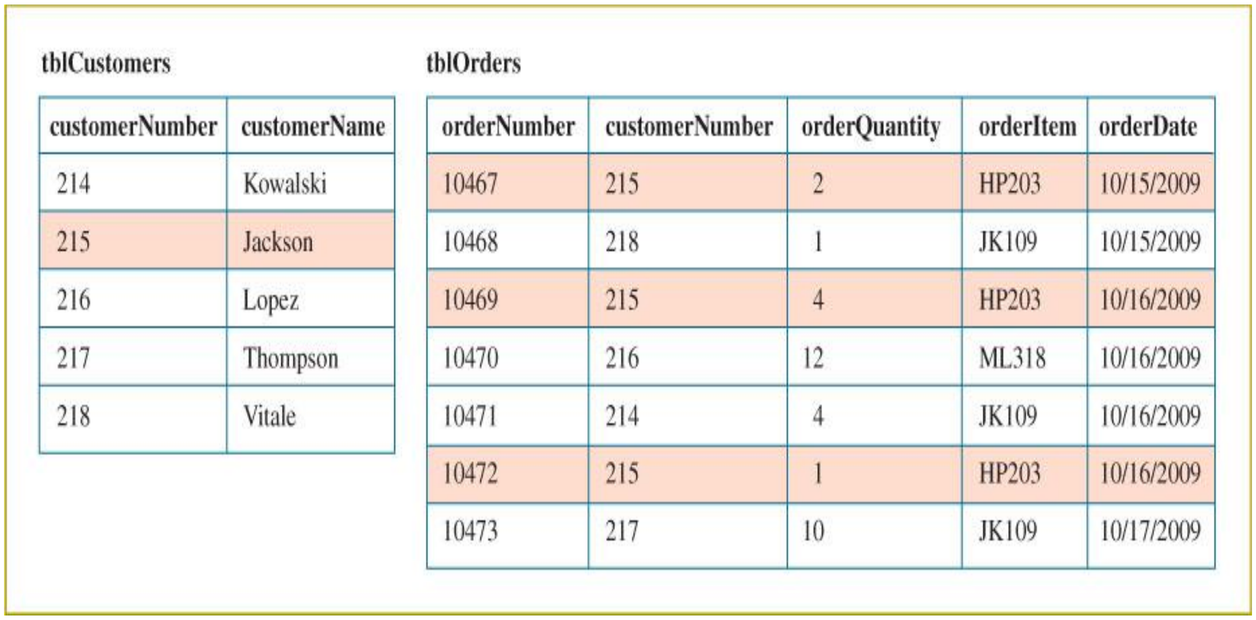


tblItems

itemNumber	itemName	itemPurchaseDate	itemPurchasePrice	itemcategoryId
1	Sofa	1/13/2003	$6,500	5
2	Stereo	2/10/2005	$1,200	6
3	Refrigerator	5/12/2005	$750	1
4	Diamond ring	2/12/2006	$42,000	2
5	TV	7/11/2006	$285	6
6	Rectangular pine coffee table	4/21/2007	$300	5
7	Round pine end table	4/21/2007	$200	5

tblCategories

categoryId	categoryName	categoryInsuredAmount
1	Appliance	$30,000
2	Jewelry	$15,000
3	Antique	$10,000
4	Clothing	$25,000
5	Furniture	$5,000
6	Electronics	$2,500
7	Miscellaneous	$5,000



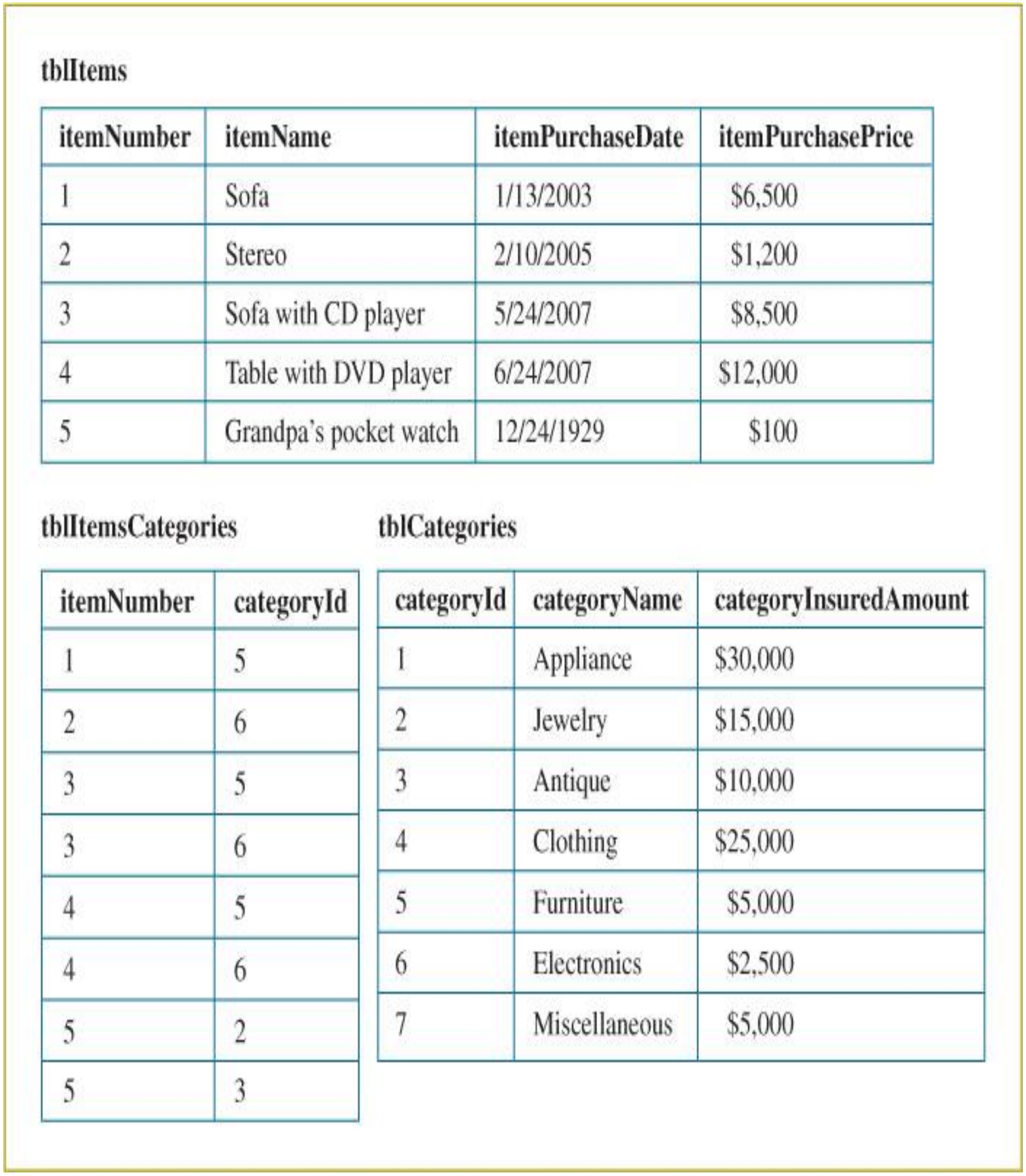
tblCustomers

customerNumber	customerName
214	Kowalski
215	Jackson
216	Lopez
217	Thompson
218	Vitale

tblOrders

orderNumber	customerNumber	orderQuantity	orderItem	orderDate
10467	215	2	HP203	10/15/2009
10468	218	1	JK109	10/15/2009
10469	215	4	HP203	10/16/2009
10470	216	12	ML318	10/16/2009
10471	214	4	JK109	10/16/2009
10472	215	1	HP203	10/16/2009
10473	217	10	JK109	10/17/2009

2.11 Many to Many



tblItems

itemNumber	itemName	itemPurchaseDate	itemPurchasePrice
1	Sofa	1/13/2003	$6,500
2	Stereo	2/10/2005	$1,200
3	Sofa with CD player	5/24/2007	$8,500
4	Table with DVD player	6/24/2007	$12,000
5	Grandpa's pocket watch	12/24/1929	$100

tblItemsCategories

itemNumber	categoryId
1	5
2	6
3	5
3	6
4	5
4	6
5	2
5	3

tblCategories

categoryId	categoryName	categoryInsuredAmount
1	Appliance	$30,000
2	Jewelry	$15,000
3	Antique	$10,000
4	Clothing	$25,000
5	Furniture	$5,000
6	Electronics	$2,500
7	Miscellaneous	$5,000

2.12 Normalization example

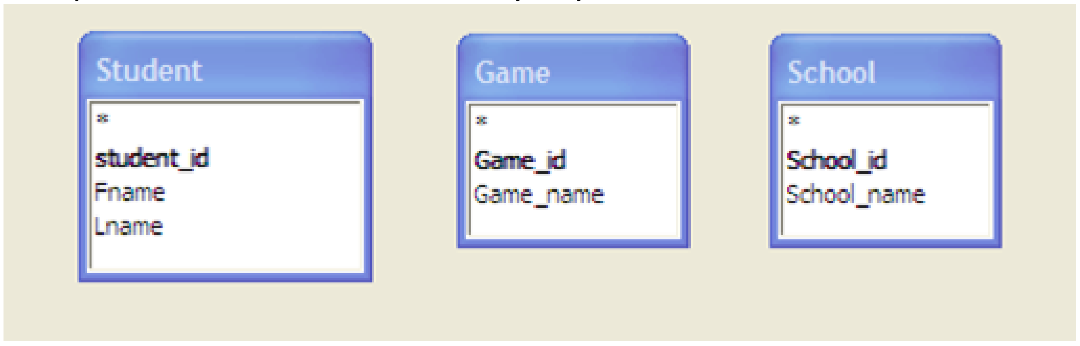
<u>Statement of the Project:</u>

"We want to store information about a bunch of **students** who play different kinds of **games**. In addition, we want to store the name of the **school** that they attend."

We make the following assumptions:

- Each student can play many games.
- Each student attends only one school.

First Step: Create Tables with Primary keys



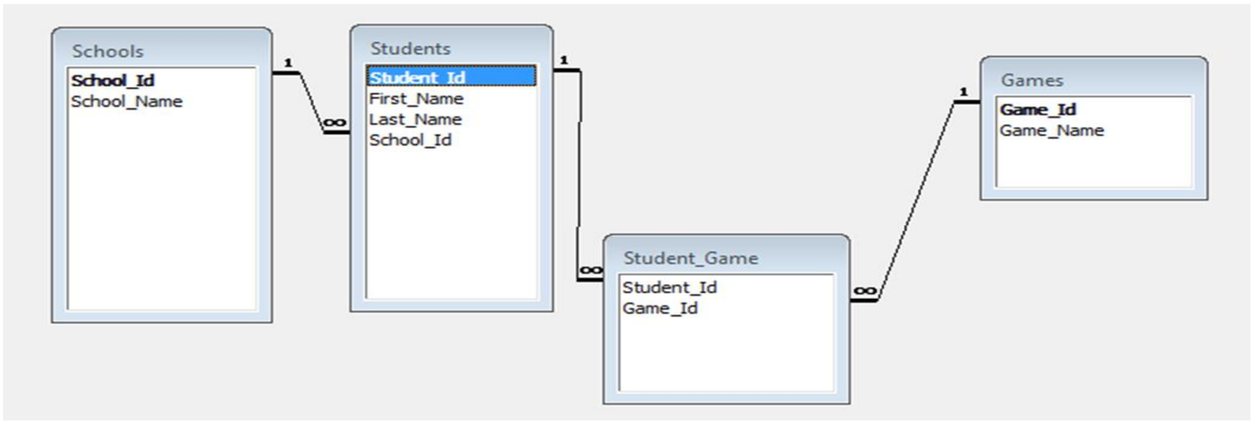
Second Step: Add Bridge table between Student and Game

- To construct a bridge between the Students and Games table, we must answer the following:
  - Can a Student play many games? Can a game be played by several students? YES
  - If the answer is yes, we have a many to many relationship.
  - This means that a bridge table is required between the Student and Game tables.

Third Step: Relationship between Student and School

- Do we need a bridge between the School and Student tables?
  - Can a school be attended by several students? Yes
  - Can a student attend different Schools? No, by assumption
  - Here we have one to man relationship. No need for bridge table

Normalized database



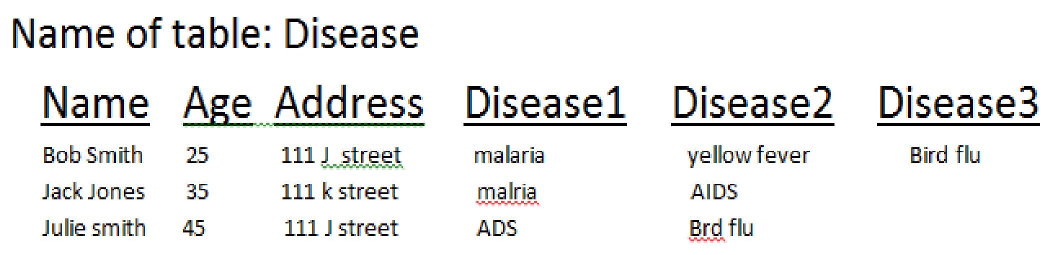
Primary and Foreign Keys:

- The primary keys:
  - School table –School_id
  - Student table – Student_id
  - Game table – Game_id
- Foreign Keys:
  - Students table – School_id
  - Student_game - Student_id, Game_id
- Candidate keys

  A possible candidate key can be a combination of last name and first name in the student table, assuming no two students have the same first name and last name. This is a bad assumption and is only used to illustrate a composite candidate key.

2.13 Normalize the following

Problem 1:



Name of table: Disease

Name	Age	Address	Disease1	Disease2	Disease3
Bob Smith	25	111 J street	malaria	yellow fever	Bird flu
Jack Jones	35	111 k street	malria	AIDS	
Julie smith	45	111 J street	ADS	Brd flu	

Problem 2

Name of table: Biography

Student	Religion	Ethn1	Ethn2
John Doe	Buddhist	White	
Jack smith	Christian	Black	white
Tiger Woods	Moslem	Asian	

Note: Tiger woods identifies himself as Cabinasian (Caucasian, Black, Indian, Asian)

✓ *CHECK 2A*

1. What is normalization?
2. What are the different normal forms?
3. What are the different types of relationships between tables?
4. What is the difference between a primary key, unique key and foreign key?
5. How many primary keys and unique keys can a table have?
6. What is a bridge table?
7. Create a normalized database that can store different people's personality type. Here is a sample of the denormalized data:

SSN	Fname	Lname	Salary	DOB	Personality type
111	John	Germs	100000	1/1/1990	Good
112	Jill	Fumbles	50000	2/1/91	Bad
113	James	Grapes	200000	3/1/88	Bad
114	Jack	Fickle	900000	3/2/77	Ugly

"True friendship is like sound health; the value of it is seldom known until it is lost."

Larry Ellison has been married and divorced several times. He was married to Adda Quinn from 1967 to 1974. He was married to Nancy Wheeler Jenkins between 1977 and 1978. From 1983 to 1986, he was married to Barbara Boothe: two children were born of this marriage, a son and daughter named David and Megan. On 18 December 2003, Ellison married Melanie Craft, a romance novelist, at his Woodside estate. His friend Steve Jobs (CEO of Apple, Inc) was the official wedding photographer and Representative Tom Lantos officiated. Ellison and Melanie Craft-Ellison divorced in September 2010.

Chapter 3 (DDL commands)

"People never grow up; they just learn how to act in public."

3.1 Creating and dropping tables

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.
- A user cannot have two tables with the same name.
- Identifiers cannot be more than 30 characters long.
- A table can have up to 1000 columns

Basic syntax

```
CREATE TABLE tablename
(
        Columnname   TYPE,
        Columnname   TYPE,
        Columnname   TYPE
);
```

Some common types:

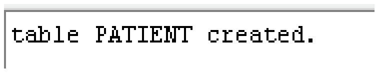
VARCHAR2(n)	Variable-length character data where **n** represents the column's maximum length. The maximum size is 4000 characters.
CHAR(n)	Fixed-length character columns where **n** represents the column's length. The default Is 1 and the maximum size is 2000.
NUMBER(p,s)	Numeric column where **p** 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.

NOTE: There are other datatypes such BINARY_FLOAT, BINARY_DOUBLE, INTEGER, LONG, CLOB, RAW, LONG RAW, BLOB, BFILE, TIMESTAMP and INTERVAL.

Example 3.1a (Create table)

Create a new, empty table called patient

```
CREATE TABLE patient
(
  patient_id        NUMBER,
  fname             VARCHAR2(20),
  lname             VARCHAR2(30),
  gender            CHAR,
  DOB               DATE,
  annual_salary     NUMBER
);
```

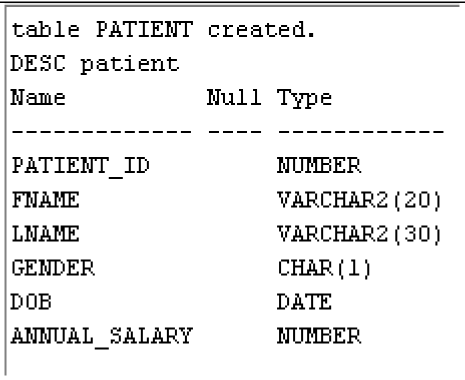


```
table PATIENT created.
```

Example 3.1b (DESC command)

DESC provides information about the columns in a table. The columns appear on the left hand side and their datatypes on the right hand side. The Heading NULL will be discussed later. A semicolon is not required for the DESC command.

```
DESC patient;
```

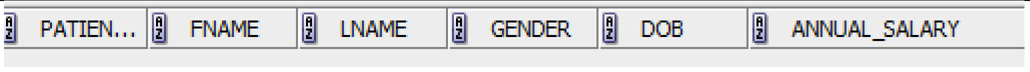


```
table PATIENT created.
DESC patient
Name          Null Type
------------- ---- ------------
PATIENT_ID         NUMBER
FNAME              VARCHAR2(20)
LNAME              VARCHAR2(30)
GENDER             CHAR(1)
DOB                DATE
ANNUAL_SALARY      NUMBER
```

Example 3.1c (View contents of table)

Looking at the contents of the table. Currently there are no rows in the patient table.

```
SELECT * FROM patient;
```

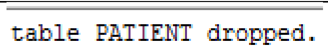


PATIEN...	FNAME	LNAME	GENDER	DOB	ANNUAL_SALARY

Example 3.1d (Drop table command)

Use the DROP TABLE statement to move a table to the recycle bin. All the data inside the table is erased.

```
Drop TABLE patient;
```

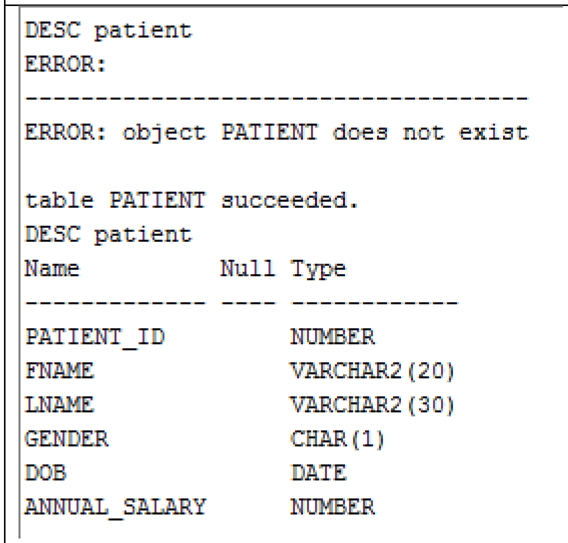


```
table PATIENT dropped.
```

Example 3.1e (Flashback)

A dropped table which is moved to the recycle bin can be recovered using FLASHBACK TABLE command.

```
--Table does not exist since it was dropped earlier.
DESC patient;


--Restore the table

FLASHBACK TABLE patient TO BEFORE DROP;



--Confirm that the table has been restored

DESC patient;
```

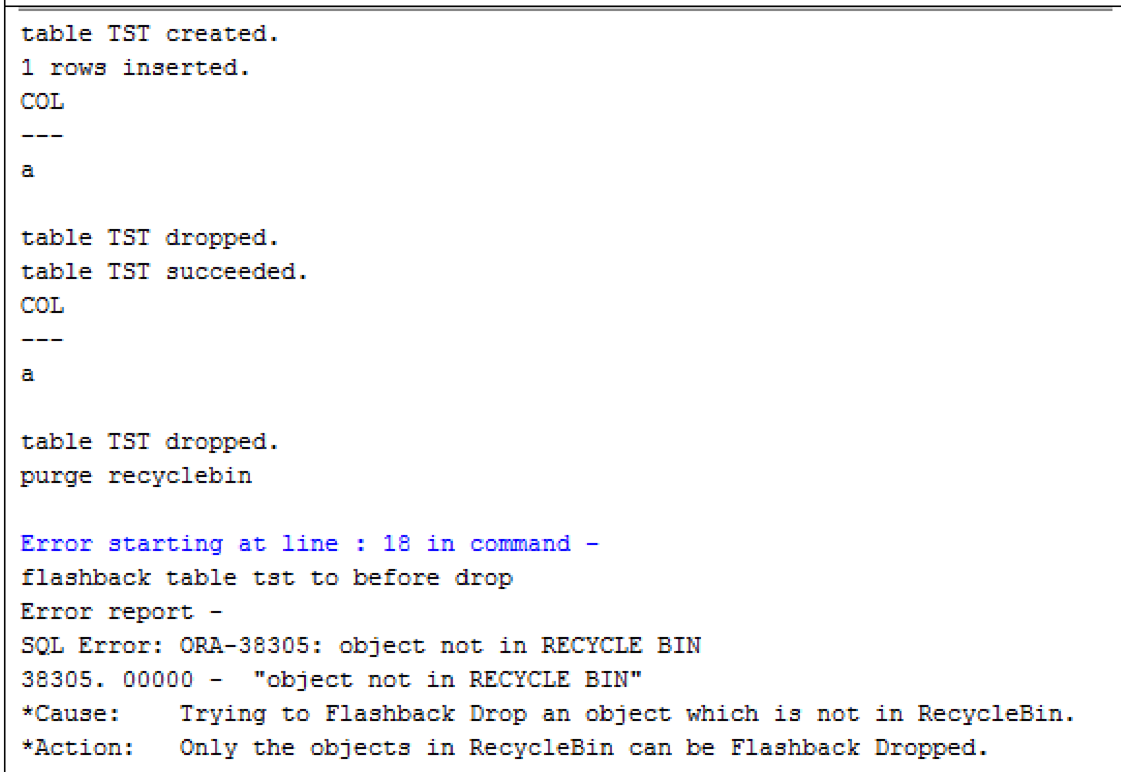


```
DESC patient
ERROR:
-------------------------------------
ERROR: object PATIENT does not exist

table PATIENT succeeded.
DESC patient
Name          Null Type
------------- ---- ------------
PATIENT_ID         NUMBER
FNAME              VARCHAR2(20)
LNAME              VARCHAR2(30)
GENDER             CHAR(1)
DOB                DATE
ANNUAL_SALARY      NUMBER
```

Example 3.1f (Recyclebin)

Purging from the recyclebin

```
CREATE TABLE tst

(
  col CHAR
);


INSERT INTO tst VALUES ('a');         --Insert a row into the table
SELECT * FROM tst;                    --Examine contents
DROP TABLE tst;                       --Remove table


FLASHBACK TABLE tst TO BEFORE DROP;   --Recover table


SELECT * FROM tst;                    --Confirm recovery
DROP TABLE tst;
PURGE RECYCLEBIN;               --Recovery is not possible after this
line
FLASHBACK TABLE tst TO BEFORE DROP;    --Error:  Nothing to recover
```

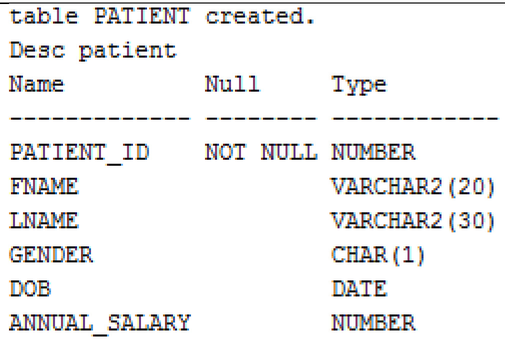


```
table TST created.
1 rows inserted.
COL
---
a

table TST dropped.
table TST succeeded.
COL
---
a

table TST dropped.
purge recyclebin

Error starting at line : 18 in command -
flashback table tst to before drop
Error report -
SQL Error: ORA-38305: object not in RECYCLE BIN
38305. 00000 -  "object not in RECYCLE BIN"
*Cause:    Trying to Flashback Drop an object which is not in RecycleBin.
*Action:   Only the objects in RecycleBin can be Flashback Dropped.
```

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.

```
CREATE TABLE patient
(
  patient_id       NUMBER NOT NULL,
  fname            VARCHAR2(20),
  lname            VARCHAR2(30),
  gender           CHAR  DEFAULT   'm',
  DOB              DATE,
  annual_salary    NUMBER
);


Desc patient;
```

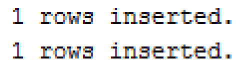


```
table PATIENT created.
Desc patient
Name          Null     Type
------------- -------- ------------
PATIENT_ID    NOT NULL NUMBER
FNAME                  VARCHAR2(20)
LNAME                  VARCHAR2(30)
GENDER                 CHAR(1)
DOB                    DATE
ANNUAL_SALARY          NUMBER
```

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.

```
INSERT INTO patient VALUES (11,'John', 'Smith', 'm','01-FEB-1970',
55000);
INSERT INTO patient VALUES (12, 'Jill', 'Doe', 'f','20-FEB-1970',
95000);
```



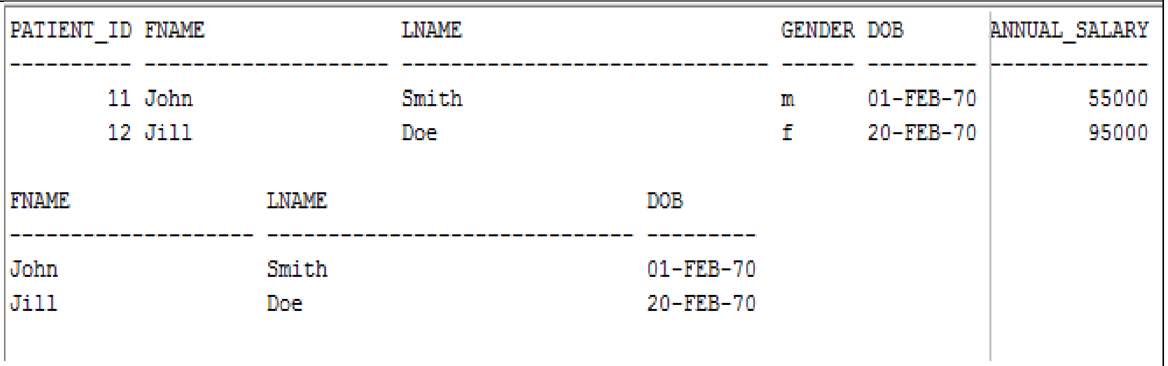
```
1 rows inserted.
1 rows inserted.
```

Example 3.1i (View contents)

Look at the contents of the table

```
--The asterisk shows all the columns for each of the two rows.
```

```
SELECT * FROM patient;


--This statement only shows the three columns for each of the two rows.
SELECT fname, lname, DOB FROM patient;
```

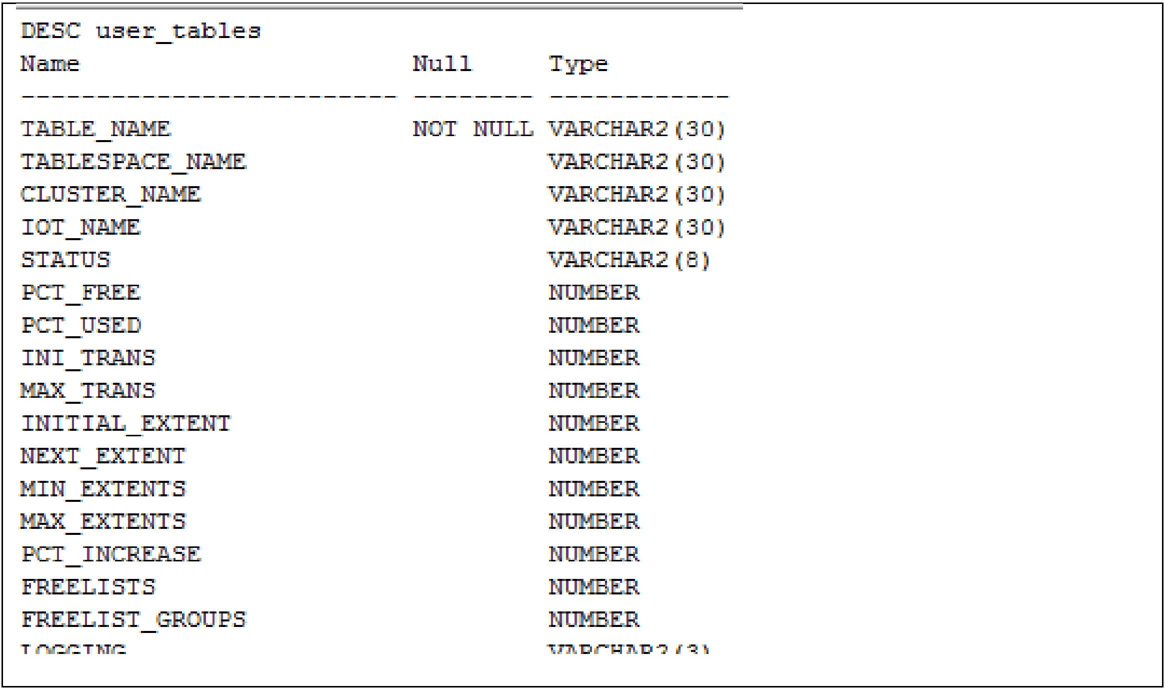


```
PATIENT_ID FNAME                LNAME                          GENDER DOB       ANNUAL_SALARY
---------- -------------------- ------------------------------ ------ --------- -------------
        11 John                 Smith                          m      01-FEB-70         55000
        12 Jill                 Doe                            f      20-FEB-70         95000

FNAME                LNAME                          DOB
-------------------- ------------------------------ ---------
John                 Smith                          01-FEB-70
Jill                 Doe                            20-FEB-70
```

Example 3.1j (Data dictionary tables: user_tables)

System tables: Oracle uses a Data Dictionary to store details of all the Tables, Columns etc..Here is an example of a row that is automatically inserted into one of the system tables (user_tables). Other system tables that we will be exploring will be dictionary, user_constraints, user_cons_columns, user_indexes, user_ind_columns.

```
--Only some of the columns are displayed.
DESC user_tables;
```



```
DESC user_tables
Name                          Null     Type
----------------------------- -------- ------------
TABLE_NAME                    NOT NULL VARCHAR2(30)
TABLESPACE_NAME                        VARCHAR2(30)
CLUSTER_NAME                           VARCHAR2(30)
IOT_NAME                               VARCHAR2(30)
STATUS                                 VARCHAR2(8)
PCT_FREE                               NUMBER
PCT_USED                               NUMBER
INI_TRANS                              NUMBER
MAX_TRANS                              NUMBER
INITIAL_EXTENT                         NUMBER
NEXT_EXTENT                            NUMBER
MIN_EXTENTS                            NUMBER
MAX_EXTENTS                            NUMBER
PCT_INCREASE                           NUMBER
FREELISTS                              NUMBER
FREELIST_GROUPS                        NUMBER
LOGGING                                VARCHAR2(3)
```

```
--All the tables that have been created by the user that is logged in are displayed.
SELECT table_name FROM user_tables;
```

```
TABLE_NAME
------------------
DESKTOP
PATIENT
```

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.

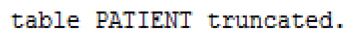
```
--Notice that there is no asterisk(*) in the delete statement.
DELETE FROM patient;
```

```
2 rows deleted.
```

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;
```



```
table PATIENT truncated.
```

Example 3.1m (Order of columns)

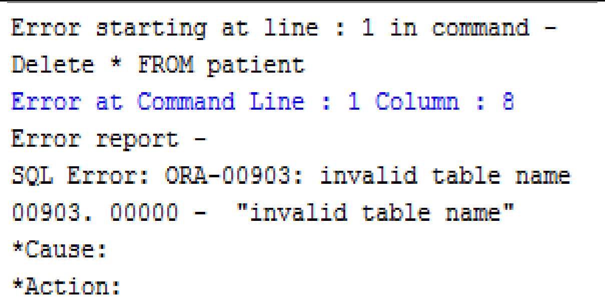
Can identify the columns in any order.

```
CREATE TABLE patient
(
    fname              VARCHAR2(20),
    lname              VARCHAR2(30),
    DOB                DATE,
    patient_id         NUMBER,
    gender             CHAR,
    annual_salary      NUMBER
);
```

Example 3.1n (Problems??)

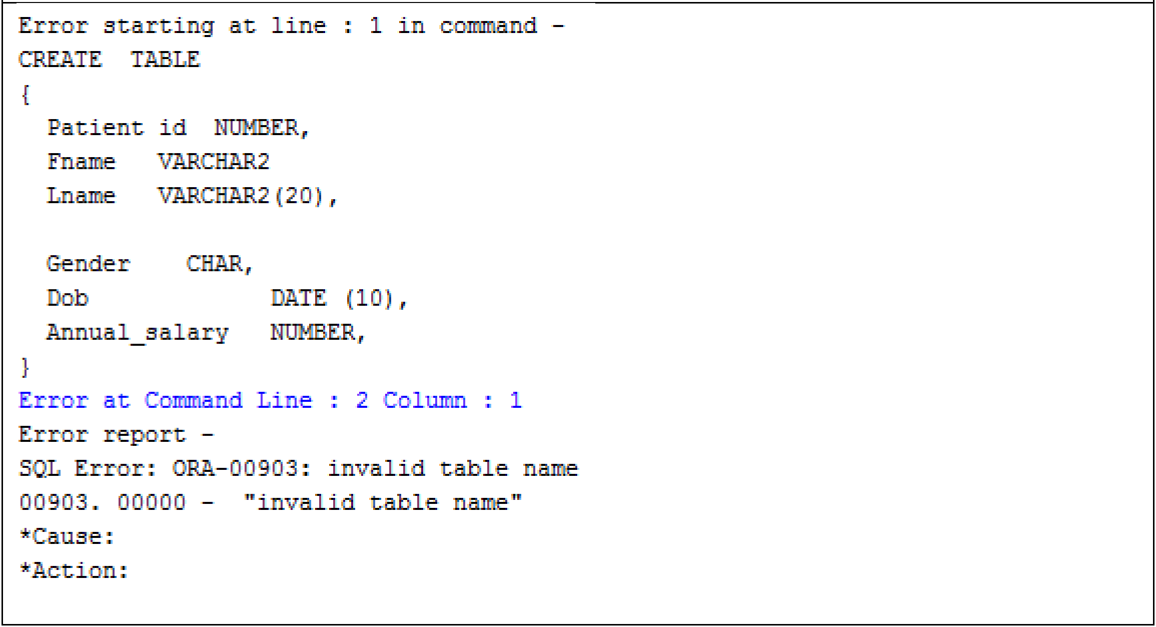
What is wrong the following?

```
--What is wrong?
Delete * FROM patient;
```



```
Error starting at line : 1 in command -
Delete * FROM patient
Error at Command Line : 1 Column : 8
Error report -
SQL Error: ORA-00903: invalid table name
00903. 00000 -  "invalid table name"
*Cause:
*Action:
```

```
CREATE  TABLE
{
      Patient id       NUMBER,
      Fname            VARCHAR2
      Lname            VARCHAR2(20),

      Gender            CHAR,
      DOB               DATE (10),
      Annual_salary     NUMBER,
};
```



```
Error starting at line : 1 in command -
CREATE  TABLE
{
  Patient id  NUMBER,
  Fname    VARCHAR2
  Lname    VARCHAR2(20),

  Gender     CHAR,
  Dob                DATE (10),
  Annual_salary    NUMBER,
}
Error at Command Line : 2 Column : 1
Error report -
SQL Error: ORA-00903: invalid table name
00903. 00000 -  "invalid table name"
*Cause:
*Action:
```

Example 3.1o (Renaming a table)

```
--Renames the table patient to sickPerson.
RENAME patient TO sickperson;
```

✓ *CHECK 3A*

1. Create a table called Person comprised of SSN, lname, fname, and salary columns.
2. Insert a record into the table and view its data.
3. Confirm the entry in the system table (USER_TABLES).
4. Delete the record.
5. Truncate the table.
6. Drop the table.
7. What is the difference between delete and truncate?
8. What is the difference between CHAR and VARCHAR?

"A quiet conscience sleeps in thunder"

3.2 Adding columns using ALTER command

At times, you need to make structural changes to a table. For example, you might need to add a column, delete a column, or simply change a column's size. Each of these changes is made with the ALTER TABLE command.

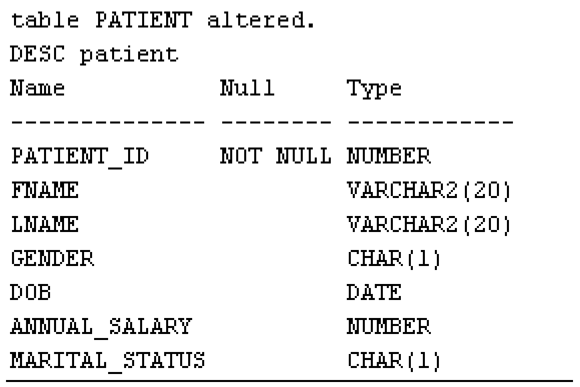
ALTER TABLE tablename ADD|MODIFY|DROP| columnname (definition);

Using an ADD clause with the ALTER TABLE command allows a user to add a new column to a table. The same rules for creating a column in a new table apply to adding a column to an existing table. The new column must be defined by a column name and datatype (and width, if applicable). A default value can also be assigned. The difference is that the new column is added at the end of the existing table— it will be the last column.

Example 3.2a (Additional column)

Add an additional column of type char

```
--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;
```

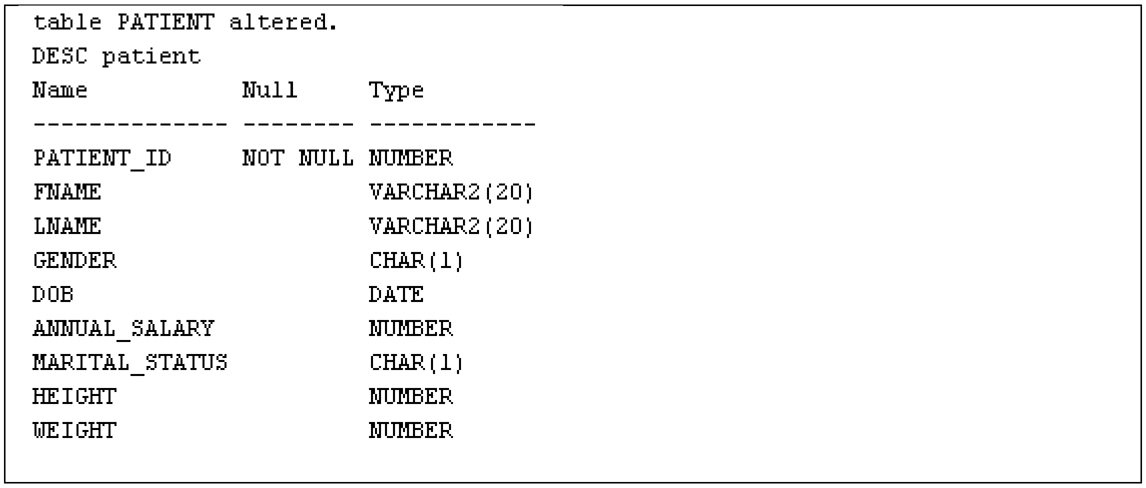


```
table PATIENT altered.
DESC patient
Name           Null     Type
-------------- -------- ------------
PATIENT_ID     NOT NULL NUMBER
FNAME                   VARCHAR2(20)
LNAME                   VARCHAR2(20)
GENDER                  CHAR(1)
DOB                     DATE
ANNUAL_SALARY           NUMBER
MARITAL_STATUS          CHAR(1)
```

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
);
```



```
table PATIENT altered.
DESC patient
Name           Null     Type
-------------- -------- ------------
PATIENT_ID     NOT NULL NUMBER
FNAME                   VARCHAR2(20)
LNAME                   VARCHAR2(20)
GENDER                  CHAR(1)
DOB                     DATE
ANNUAL_SALARY           NUMBER
MARITAL_STATUS          CHAR(1)
HEIGHT                  NUMBER
WEIGHT                  NUMBER
```

3.3 Modifying using ALTER command

To change an existing column's definition, you can use a MODIFY clause with the ALTER TABLE command. The changes that can be made to a column include the following:

- 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)

You should be aware of three rules when modifying existing columns:

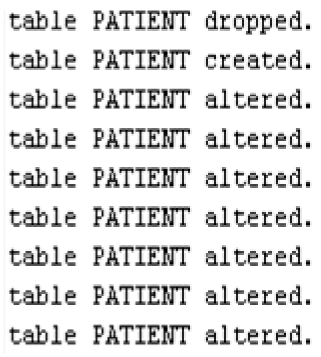
- A column must be as wide as the data fields it already contains.
- If a NUMBER column already contains data, you can't decrease the column's precision or scale.
- Changing the default value of a column doesn't change the values of data already in the table

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.

```
DROP TABLE patient;

CREATE  TABLE Patient
(
      Patient_id  NUMBER NOT NULL,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      Gender            CHAR DEFAULT 'm',
      DOB               DATE,
      Annual_salary     NUMBER
);
```

```
--We are free to change the datatypes by  shortening, lengthening or even modifying
--from textual to numeric, as long as there is no data in the table. Once there is data, then there
are
-- some restrictions.
ALTER TABLE patient MODIFY fname VARCHAR2(19);
ALTER TABLE patient MODIFY fname VARCHAR2(29);
ALTER TABLE patient MODIFY fname NUMBER;
ALTER TABLE patient MODIFY fname VARCHAR2(20);
ALTER TABLE patient MODIFY patient_id CHAR;
ALTER TABLE patient MODIFY patient_id NUMBER NULL;
ALTER TABLE patient MODIFY patient_id NOT NULL;
```



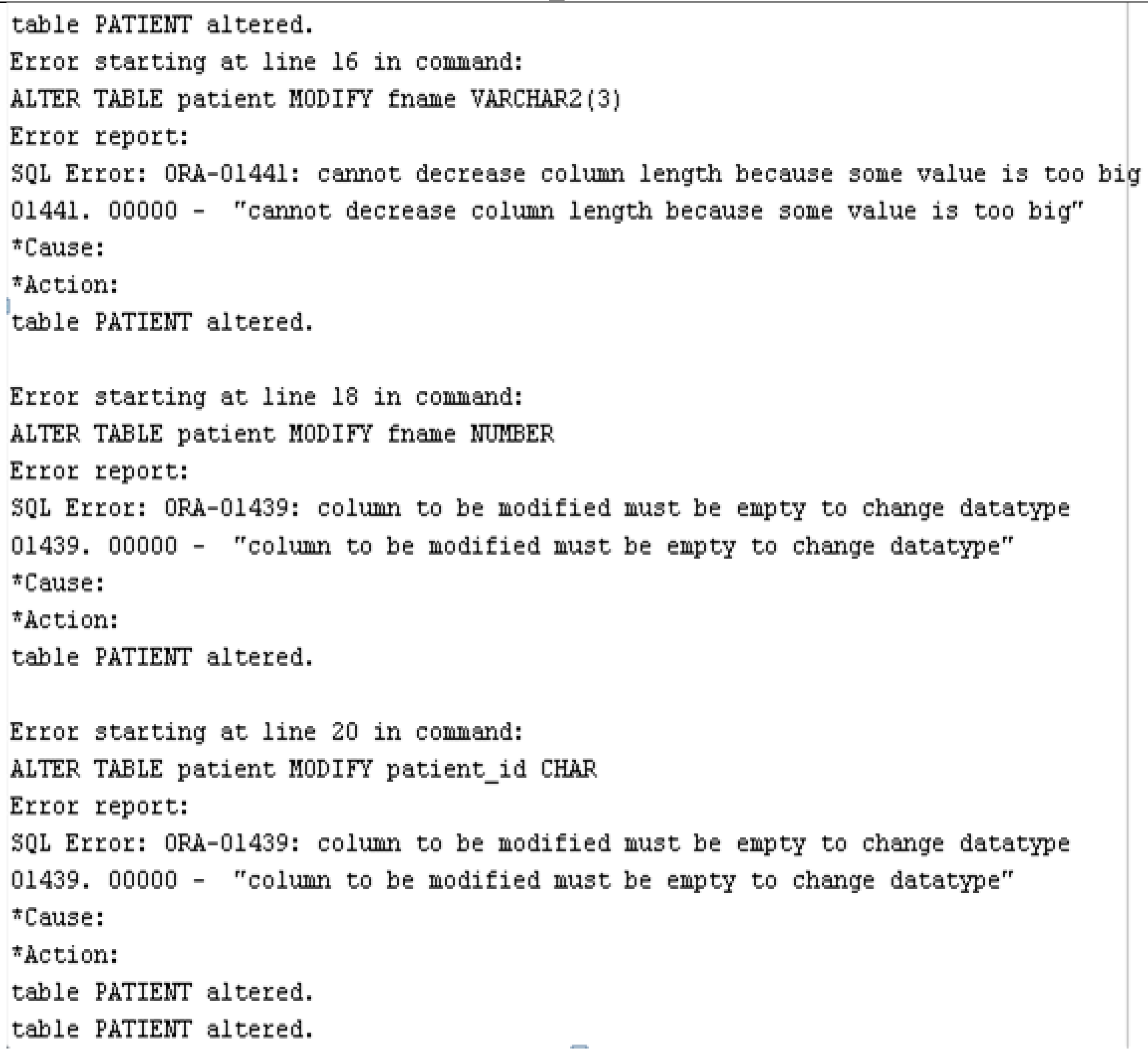
```
table PATIENT dropped.
table PATIENT created.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
```

Example 3.3b (Restictions on modify)

The modify command is more restrictive in terms of what can and cannot be done if the table is not empty

```
DROP TABLE patient;

CREATE  TABLE Patient
(
      Patient_id          NUMBER NOT NULL,
      Fname               VARCHAR2(20),
      Lname               VARCHAR2(20),
      Gender              CHAR DEFAULT 'm',
      DOB                 DATE,
      Annual_salary       NUMBER
);

INSERT INTO patient VALUES (11,'John', 'Smith', 'm','01-FEB-1970', 55000);
INSERT INTO patient VALUES (12, 'Jill', 'Doe', 'f','20-FEB-1970', 95000);
```

```
--Only the results from this point forward are displayed below.
ALTER TABLE patient MODIFY fname VARCHAR2(19);

--INVALID: The length is too short because it truncates existing data.
ALTER TABLE patient MODIFY fname VARCHAR2(3);

ALTER TABLE patient MODIFY fname VARCHAR2(29);        --valid

--INVALID: Datatype does not match existing data.
ALTER TABLE patient MODIFY fname NUMBER;

ALTER TABLE patient MODIFY fname VARCHAR2(20);        --valid

--INVALID: Data type does not match.
ALTER TABLE patient MODIFY patient_id CHAR;

ALTER TABLE patient MODIFY patient_id NUMBER NULL;   --valid
ALTER TABLE patient MODIFY patient_id NOT NULL;       --valid
```



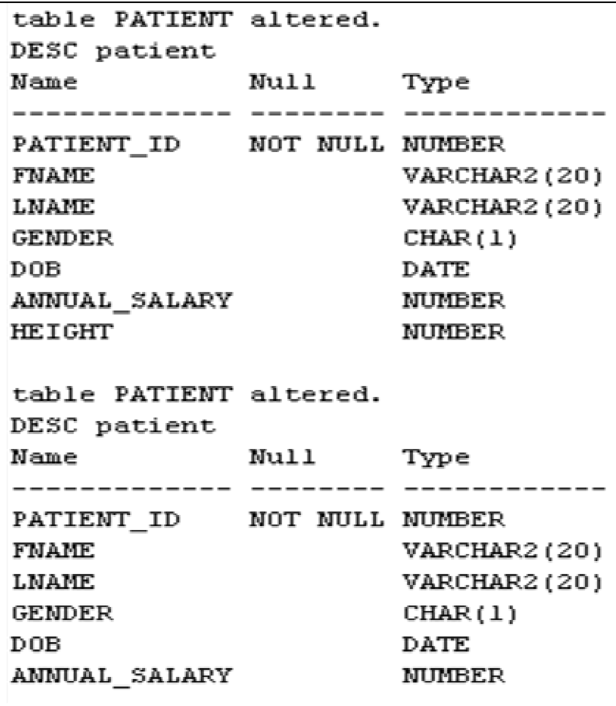
```
table PATIENT altered.
Error starting at line 16 in command:
ALTER TABLE patient MODIFY fname VARCHAR2(3)
Error report:
SQL Error: ORA-01441: cannot decrease column length because some value is too big
01441. 00000 -  "cannot decrease column length because some value is too big"
*Cause:
*Action:
table PATIENT altered.

Error starting at line 18 in command:
ALTER TABLE patient MODIFY fname NUMBER
Error report:
SQL Error: ORA-01439: column to be modified must be empty to change datatype
01439. 00000 -  "column to be modified must be empty to change datatype"
*Cause:
*Action:
table PATIENT altered.

Error starting at line 20 in command:
ALTER TABLE patient MODIFY patient_id CHAR
Error report:
SQL Error: ORA-01439: column to be modified must be empty to change datatype
01439. 00000 -  "column to be modified must be empty to change datatype"
*Cause:
*Action:
table PATIENT altered.
table PATIENT altered.
```

3.4 Dropping columns using ALTER command

Example 3.4a (Dropping column)

Getting rid of a column (Cannot be rolled back)

```
--Add the column
ALTER TABLE patient ADD  height  NUMBER;

DESC patient;

--Get rid of the column and all the data in it
ALTER TABLE patient  DROP (height);

DESC patient;
```

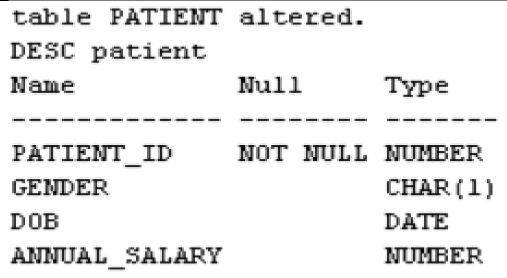


```
table PATIENT altered.
DESC patient
Name           Null     Type
-------------- -------- ------------
PATIENT_ID     NOT NULL NUMBER
FNAME                   VARCHAR2(20)
LNAME                   VARCHAR2(20)
GENDER                  CHAR(1)
DOB                     DATE
ANNUAL_SALARY           NUMBER
HEIGHT                  NUMBER

table PATIENT altered.
DESC patient
Name           Null     Type
-------------- -------- ------------
PATIENT_ID     NOT NULL NUMBER
FNAME                   VARCHAR2(20)
LNAME                   VARCHAR2(20)
GENDER                  CHAR(1)
DOB                     DATE
ANNUAL_SALARY           NUMBER
```

Example 3.4b (Dropping columns)

Getting rid of multiple columns.

```
--Use a comma to separate the columns that are to be dropped. All the data along with the columns
--will be discarded.
ALTER TABLE patient  DROP (annual_salary, marital_status);

DESC patient;
```

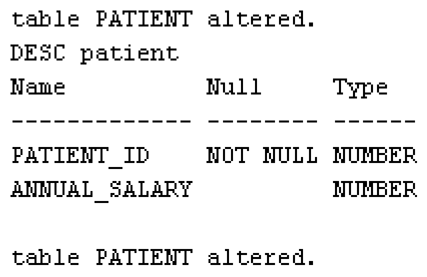


```
table PATIENT altered.
DESC patient
Name          Null     Type
------------- -------- -------
PATIENT_ID    NOT NULL NUMBER
GENDER                 CHAR(1)
DOB                    DATE
ANNUAL_SALARY          NUMBER
```

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;
```



```
table PATIENT altered.
DESC patient
Name          Null     Type
------------- -------- ------
PATIENT_ID    NOT NULL NUMBER
ANNUAL_SALARY          NUMBER

table PATIENT altered.
```

3.5 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;
```

✓ *CHECK 3B*

1. Add the column DOB (type char(10)) to the Person table using the ALTER syntax.
2. Modify the datatype to date.
3. Drop the column fname.

4. Set the DOB column to unused and then drop all unused columns.
5. Rename the lname column to fname using the ALTER syntax.

"I can't change the direction of the wind, but I can adjust my sails to always reach my destination. "

3.6 Constraints

Constraints are rules used to enforce business rules, practices, and policies to ensure the accuracy and integrity of data.You can add constraints during table creation as part of the CREATE TABLE command, or you can do so after the table is created by using the ALTER TABLE command.

When creating a table, you can create a constraint in two ways: at the column level or the table level. Creating a 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.

You can create any type of constraint at the column level— unless the constraint is being defined for more than one column (for example, a composite primary key). If the constraint applies to more than one column, you must create the constraint at the table level. Also, a NULL and DEFAULT constraint can only be created at the column level.

PRIMARY KEY	Determines which column(s) uniquely identifies each record. The primary key cannot be NULL and the data values must be unique
FOREIGN KEY	In one to many relationships, the constraint is added to the "many" table. The constraint ensures that if a value is entered in a specified column, it must already exist in the "One" table, or the record isn't added
UNIQUE	Ensures that all data values stored in specified column are unique. The UNIQUE constraint differs from the PRIMARY KEY constraint in that it allows NULL values.
CHECK	Ensures that a specified condition is true before the data value is added to a table. For example, an order's ship date can't be earlier than its order date.
NOT NULL	Ensures that a specified column can't contain a NULL value. The NOT NULL constraint can be created only with the column level approach to the table creation.

When creating a constraint, you can name the constraint or omit the constraint name and allow Oracle to generate the name. If the Oracle names the constraint, it follows the format SYS_ Cn, where n is an assigned numeric value to make the name unique. Providing a descriptive name for a constraint is the preferred practice so that you can identify it easily in the future. For example, constraint violation errors reference the constraint name, so an easy- to- understand name indicating the table, column, and type of constraint is quite helpful. Industry convention is to use the format tablename_ columnname_ constrainttype for the constraint name— for example, patient_ patient#_ pk. Constraint types are designated by abbreviations, as such:

Constraint	**Abbreviation**
PRIMARY KEY	_pk
FOREIGN KEY	_fk
UNIQUE	_uk
CHECK	_ck
NOT NULL	_nn

When creating a constraint using the ALTER statement, the add and the modify options can be used. A constraint name cannot be assigned with the modify option when creating a check constraint.

- A constraint is a rule applied to data being added to a table. It represents business rules, policies, or procedures. Data violating the constraint isn't added to the table.
- A constraint can be included during table creation as part of the CREATE TABLE command or added to an existing table with the ALTER TABLE command.
- A constraint based on composite columns (more than one column) must be created by using the table- level approach.
- A NOT NULL constraint can be created only with the column- level approach.
- A PRIMARY KEY constraint doesn't allow duplicate or NULL values in the designated column.
- Only one PRIMARY KEY constraint is allowed in a table.
- A FOREIGN KEY constraint requires that the column entry match a referenced column entry in the table or be NULL.

- A UNIQUE constraint is similar to a PRIMARY KEY constraint, except it allows storing NULL values in the specified column.
- A CHECK constraint ensures that data meets a given condition before it's added to the table. The condition can't reference the SYSDATE function or values stored in other rows.
- A NOT NULL constraint is a special type of CHECK constraint. If you're adding to an existing column, the ALTER TABLE ... MODIFY command must be used.
- A column can be assigned multiple constraints.
- The data dictionary views USER_ CONSTRAINTS and USER_ CONS_ COLUMNS enable you to verify existing constraints.
- A constraint can be disabled or enabled with the ALTER TABLE command and the DISABLE and ENABLE keywords.
- A constraint can't be modified. To change a constraint, you must first drop it with the DROP command and then re- create it.

When a constraint exists for a column, each entry made to that column is evaluated to determine whether the value is allowed in that column (that is, it doesn't violate the constraint). If you're adding a large block of data to a table, this validation process can severely slow down the Oracle server's processing speed. If you're certain the data you're adding adheres to the constraints, you can disable the constraints while adding that particular block of data to the table.

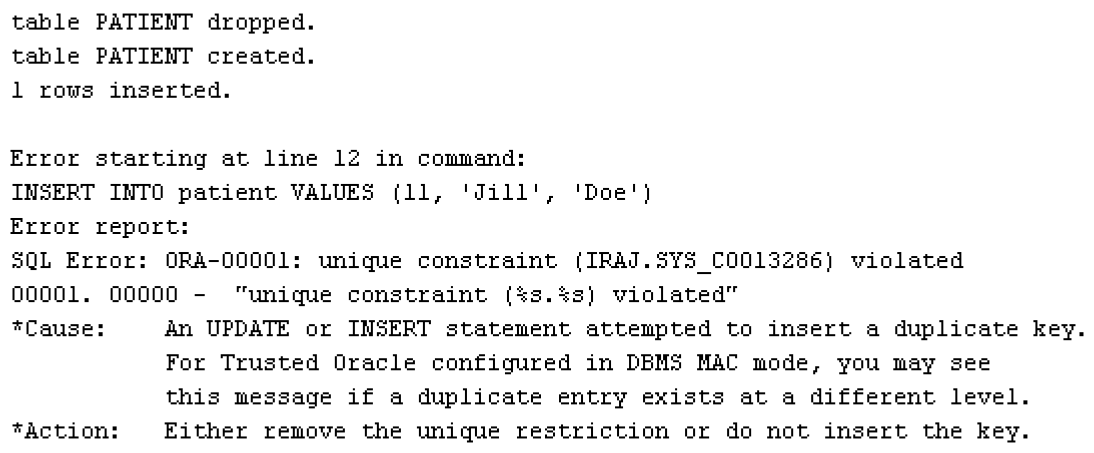
3.7 Primary key constraint

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');


--The primary key is violated because patient_id (11) is repeated.
INSERT INTO patient VALUES (11, 'Jill', 'Doe');
```

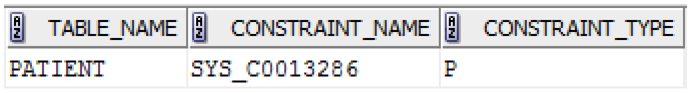


```
table PATIENT dropped.
table PATIENT created.
1 rows inserted.

Error starting at line 12 in command:
INSERT INTO patient VALUES (11, 'Jill', 'Doe')
Error report:
SQL Error: ORA-00001: unique constraint (IRAJ.SYS_C0013286) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
           this message if a duplicate entry exists at a different level.
*Action:   Either remove the unique restriction or do not insert the key.
```

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';
```



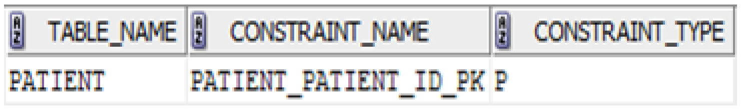
TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE
PATIENT	SYS_C0013286	P

Example 3.7c (Giving name to constraints)

Providing a descriptive name for a constraint is good practice so that you can identify it easily in the future. For example, constraint violation errors reference the constraint name, so an easy- to-understand name, indicating the table, column, and type of constraint is quite helpful.

```
DROP TABLE patient;
```

```
--Notice the use of the CONSTRAINT keyword when  giving a name to a constraint.
CREATE  TABLE Patient

(

      Patient_id  NUMBER CONSTRAINT patient_patient_id_pk PRIMARY KEY,

      Fname       VARCHAR2(20),

      Lname       VARCHAR2(20)

);



INSERT INTO patient VALUES (11,'John', 'Smith');

INSERT INTO patient VALUES (11,'Jill', 'Doe');



--Results are displayed below from this point forward.
--Notice the name of the constraint is no longer generated by the system.
SELECT table_name, constraint_name, constraint_type FROM

user_constraints WHERE table_name='PATIENT';
```



TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE
PATIENT	PATIENT_PATIENT_ID_PK	P

Example 3.7d (Primary key at the table level)

Creating a primary key constraint at the table level.

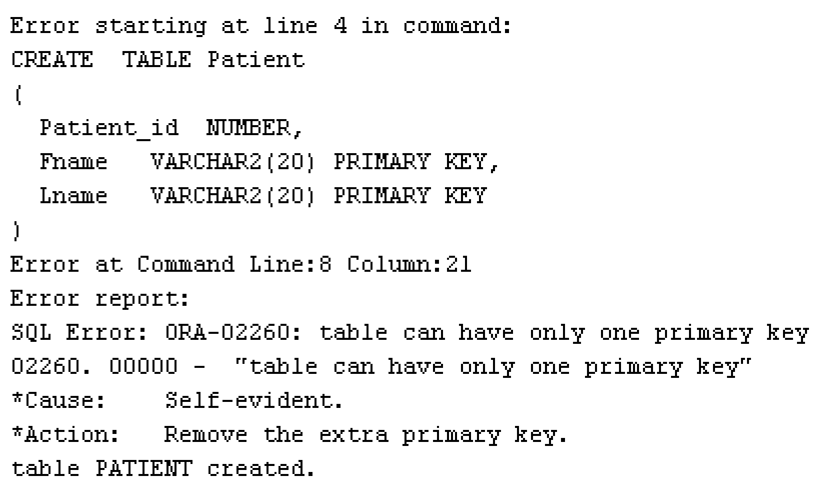
```
DROP TABLE patient;

--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;
```

```
--In this case, Oracle will generate a constraint name.
CREATE  TABLE Patient
(
      Patient_id  NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      PRIMARY KEY (patient_id)
);
```

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)
);
```



```
Error starting at line 4 in command:
CREATE  TABLE Patient
(
  Patient_id  NUMBER,
  Fname   VARCHAR2(20) PRIMARY KEY,
  Lname   VARCHAR2(20) PRIMARY KEY
)
Error at Command Line:8 Column:21
Error report:
SQL Error: ORA-02260: table can have only one primary key
02260. 00000 -  "table can have only one primary key"
*Cause:    Self-evident.
*Action:   Remove the extra primary key.
table PATIENT created.
```

Example 3.7f (Using alter table command)

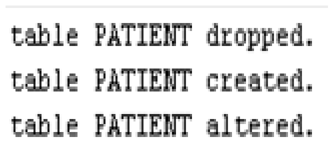
Creating a primary key using the alter command

```
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);



--Add a constraint name.
ALTER TABLE patient ADD CONSTRAINT patient_patient_id_pk PRIMARY KEY
(patient_id);



--Instead of adding, modify can be used as well.
ALTER TABLE patient MODIFY  patient_id  PRIMARY KEY;

ALTER TABLE patient MODIFY  patient_id  CHAR  PRIMARY KEY;

ALTER TABLE patient MODIFY  patient_id CONSTRAINT
patient_patient_id_pk  PRIMARY KEY;
```

Example 3.7g (Using alter table for comosite primary key)

Creating a composite primary key using the alter command

```
DROP TABLE patient;



CREATE  TABLE Patient

(
        Patient_id  NUMBER,

        Fname       VARCHAR2(20),

        Lname       VARCHAR2(20)

);



--Creating a composite primary key without a constraint name.
ALTER TABLE patient ADD PRIMARY KEY (fname, lname);
```



```
table PATIENT dropped.
table PATIENT created.
table PATIENT altered.
```

✓ *CHECK 3C*

1. What is the difference between a table and a column level constraint?
2. What happens if you don't give a name to a constraint?
3. What keyword do you use to give constraints a name?
4. What constraint do you have to create at the table level?
5. What constraint do you have to create at the column level?

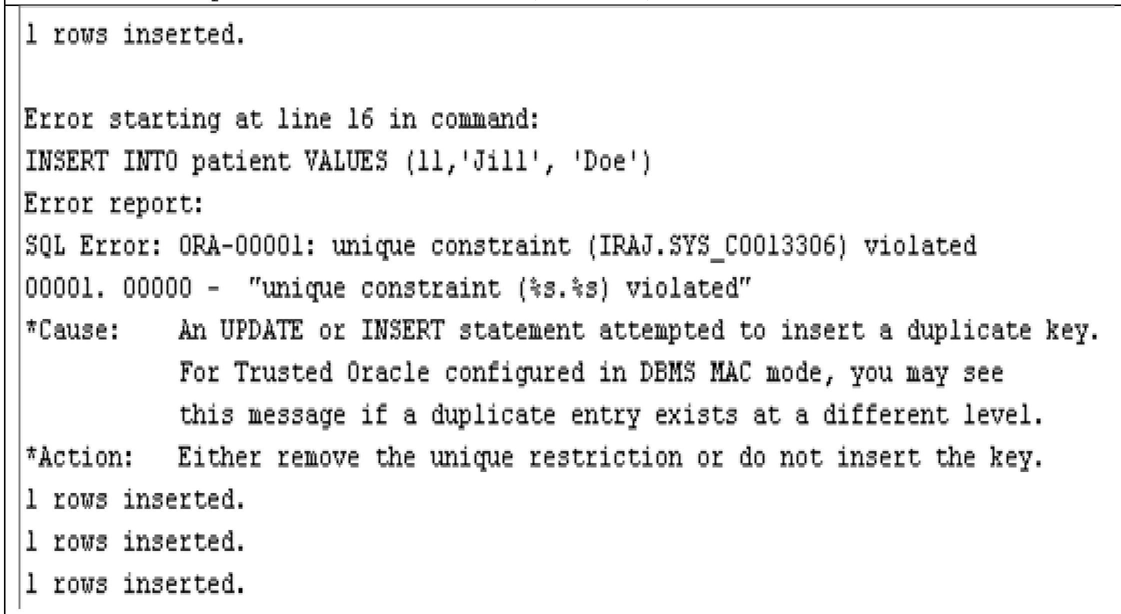
"Don't walk in front of me, I may not follow.
Don't walk behind me, I may not lead.
Just walk beside me and be my friend. "

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)
);

INSERT INTO patient VALUES (11,'John', 'Smith');

--INVALID: The patient_id (11) is a duplicate.
INSERT INTO patient VALUES (11,'Jill', 'Doe');

--NULL is allowed
INSERT INTO patient VALUES (NULL,'Jill', 'Doe');

--Since NULL means “void of data”, it is not considered a duplicate and is therefore valid.
INSERT INTO patient VALUES (NULL,'Jill', 'Doe');
INSERT INTO patient VALUES (NULL,'Jill', 'Doe');
```

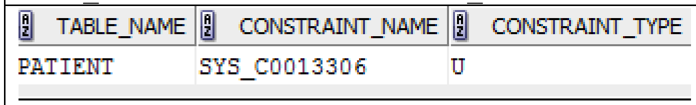


```
1 rows inserted.


Error starting at line 16 in command:
INSERT INTO patient VALUES (11,'Jill', 'Doe')
Error report:
SQL Error: ORA-00001: unique constraint (IRAJ.SYS_C0013306) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
           this message if a duplicate entry exists at a different level.
*Action:   Either remove the unique restriction or do not insert the key.
1 rows inserted.
1 rows inserted.
1 rows inserted.
```

Example 3.8b (User_constraints)

Examining the system table user_constraints

```
--The constraint name was generated by the Oracle since the constraint keyword was not used. The
--constraint type for unique key is (U).
SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';
```

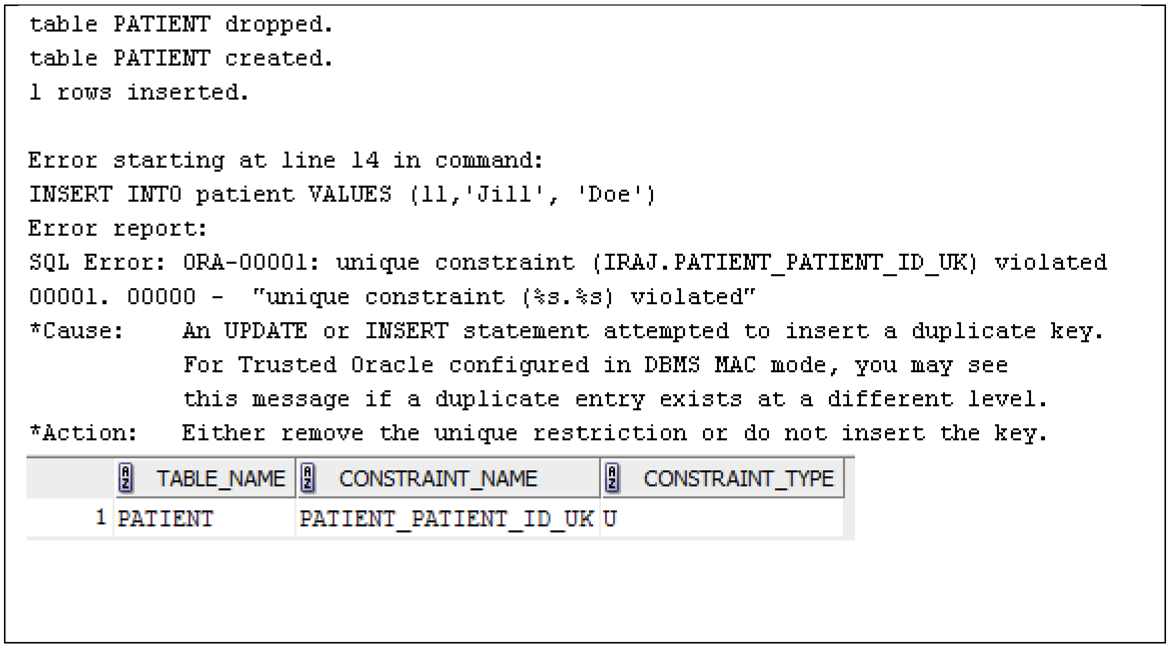


TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE
PATIENT	SYS_C0013306	U

Example 3.8c (Giving a name to constraints)

Providing a descriptive name for a constraint is a good practice so that you can identify it easily in the future. For example, constraint violation errors reference the constraint name, so an easy- to-understand name, indicating the table, column, and type of constraint is quite helpful.

```
DROP TABLE patient;



--The unique key is given a name using the constraint keyword.
CREATE  TABLE Patient

(

      Patient_id  NUMBER CONSTRAINT patient_patient_id_uk UNIQUE,

      Fname       VARCHAR2(20),

      Lname       VARCHAR2(20)

);



INSERT INTO patient VALUES (11,'John', 'Smith');



--Notice that the error message identifies the name of the constraint that is being violated.
INSERT INTO patient VALUES (11,'Jill', 'Doe');



--Notice the constraint type is (U). Also the table name must be in upper-case otherwise
--it will not be found in the ORACLE system table.
SELECT table_name, constraint_name, constraint_type FROM

user_constraints WHERE table_name='PATIENT';
```



```
table PATIENT dropped.
table PATIENT created.
1 rows inserted.

Error starting at line 14 in command:
INSERT INTO patient VALUES (11,'Jill', 'Doe')
Error report:
SQL Error: ORA-00001: unique constraint (IRAJ.PATIENT_PATIENT_ID_UK) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
           this message if a duplicate entry exists at a different level.
*Action:   Either remove the unique restriction or do not insert the key.
```

	TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	PATIENT	PATIENT_PATIENT_ID_UK	U

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)

);




DROP TABLE patient;



--The system generates a name for the unique key.
CREATE  TABLE Patient
```

```
(
      Patient_id  NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      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. Unlike the primary key constraint, we can have many unique keys. Both of the following are correct; however, in one case each individual column must be unique from row to row whereas in the other, the combination must be unique from row to row

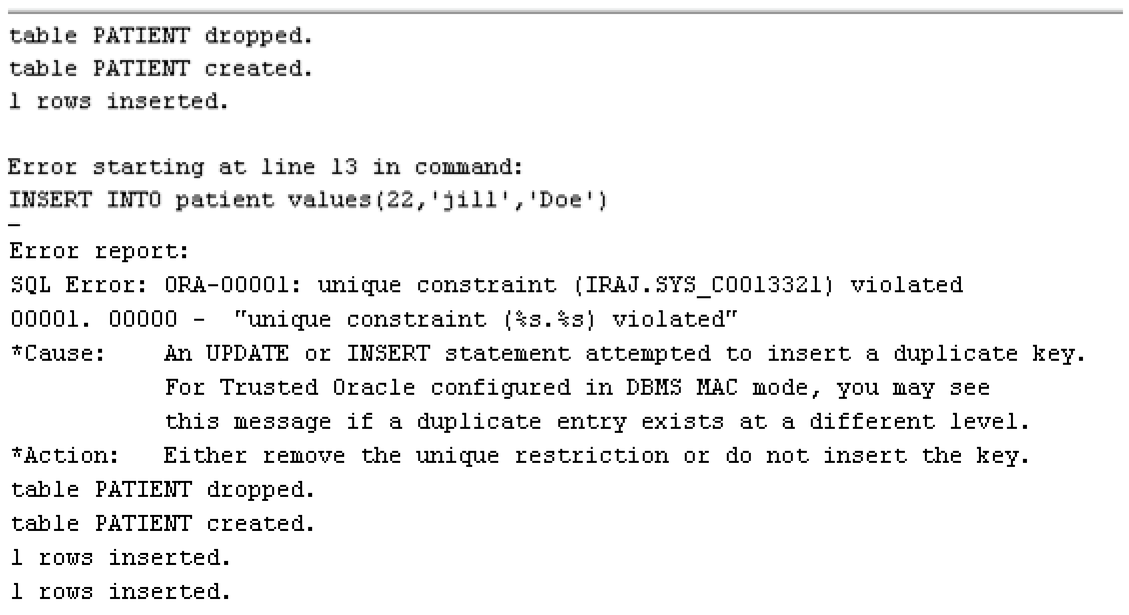
```
DROP TABLE patient;


--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
);
INSERT INTO patient VALUES (11,'John','Doe');


--The unique key is violated because of the duplicate lname (Doe)
INSERT INTO patient values(22,'jill','Doe');




DROP TABLE patient;
```

--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)

);



INSERT INTO patient VALUES (11,'John','Doe');
```

--Unique key is not violated because the combination, lname and fname, must together be unique.

```
INSERT INTO patient values(22,'jill','Doe');
```



```
table PATIENT dropped.
table PATIENT created.
1 rows inserted.

Error starting at line 13 in command:
INSERT INTO patient values(22,'jill','Doe')
-
Error report:
SQL Error: ORA-00001: unique constraint (IRAJ.SYS_C0013321) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
           this message if a duplicate entry exists at a different level.
*Action:   Either remove the unique restriction or do not insert the key.
table PATIENT dropped.
table PATIENT created.
1 rows inserted.
1 rows inserted.
```

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);
```

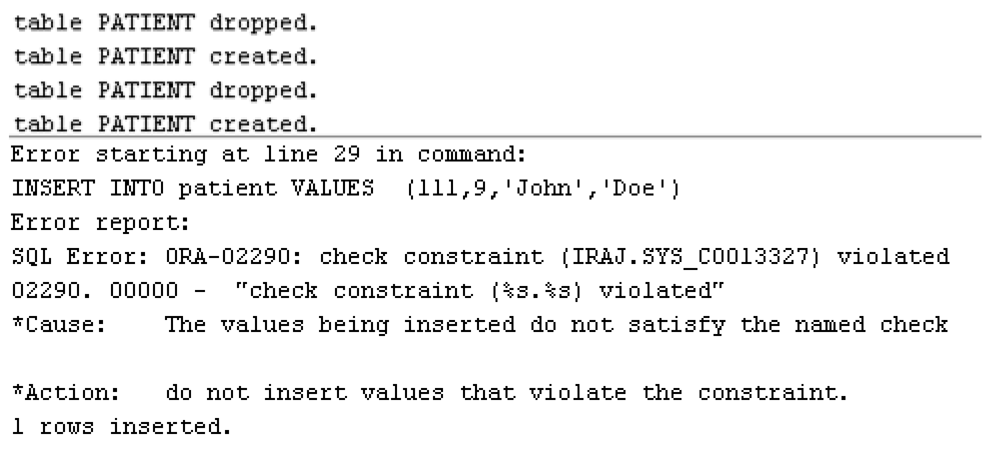
3.9 Check constraint

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)
);
```

```
DROP TABLE patient;

--This check constraint is being created at the table level. Notice all the columns are defined fully and
--the check constraint appears at the very end. The check constraint does not have a name.
CREATE  TABLE Patient
(
      Patient_id  NUMBER,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      CHECK (height>10)
);

--Violates the check constraint because of the height. Must be greater than 10.
INSERT INTO patient VALUES  (111,9,'John','Doe');

--This is okay
INSERT INTO patient VALUES  (111,19,'John','Doe');
```

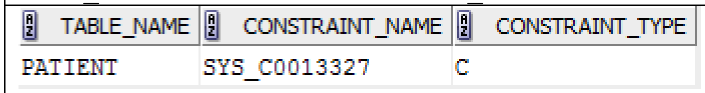


```
table PATIENT dropped.
table PATIENT created.
table PATIENT dropped.
table PATIENT created.
Error starting at line 29 in command:
INSERT INTO patient VALUES  (111,9,'John','Doe')
Error report:
SQL Error: ORA-02290: check constraint (IRAJ.SYS_C0013327) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
1 rows inserted.
```

Example 3.9b (User_constraints)

Examining the check constraint in the system table

```
--Constraint type is (c), which stands for check. Note, table name must be in UPPER CASE.
SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';
```

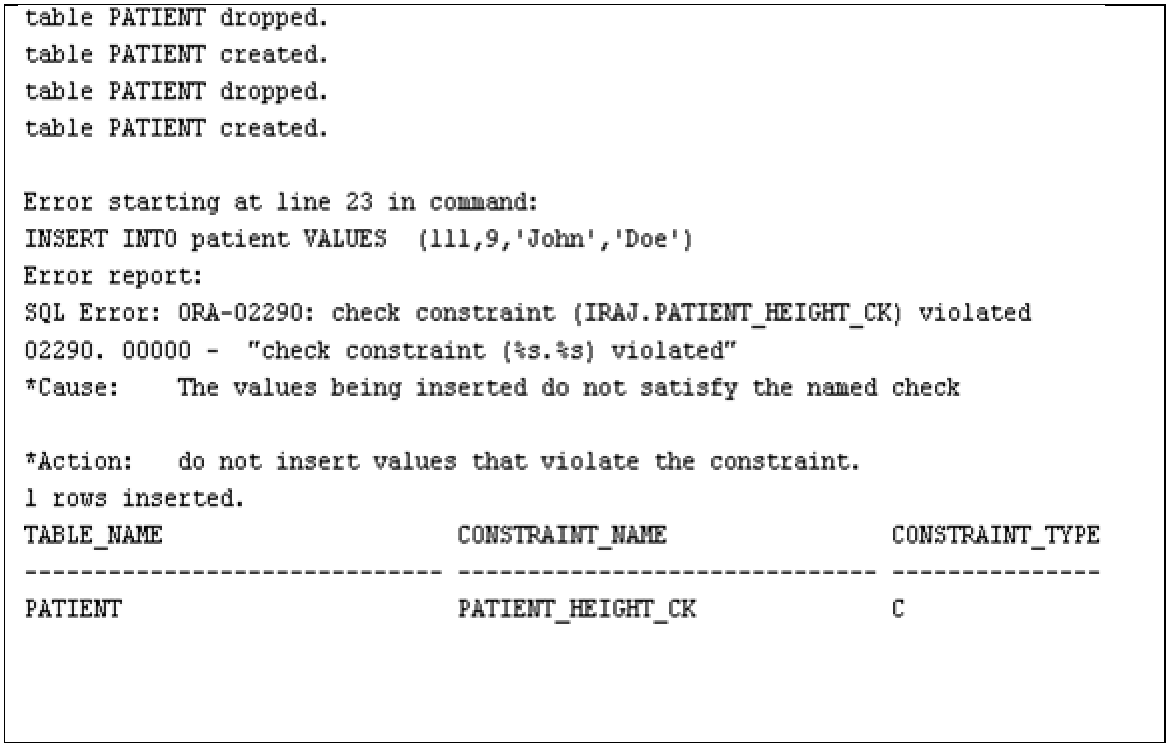


TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE
PATIENT	SYS_C0013327	C

Example 3.9c (Constraint name for table versus column level)

Column level versus table level with a constraint name

```
DROP TABLE patient;



--The check constraint, created at the column level, is given a name using the CONSTRAINT
keyword.
CREATE  TABLE Patient
(
      Patient_id  NUMBER,
      Height      NUMBER  CONSTRAINT patient_height_ck CHECK
height>10),
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);
DROP TABLE patient;

--This check constraint is created at the table level.
CREATE  TABLE Patient
(
      Patient_id  NUMBER,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      CONSTRAINT patient_height_ck  CHECK (height>10)
);



--Check constraint is violated.
INSERT INTO patient VALUES  (111,9,'John','Doe');
INSERT INTO patient VALUES (111,19,'John','Doe');

--Examine the constraint name.
SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';
```

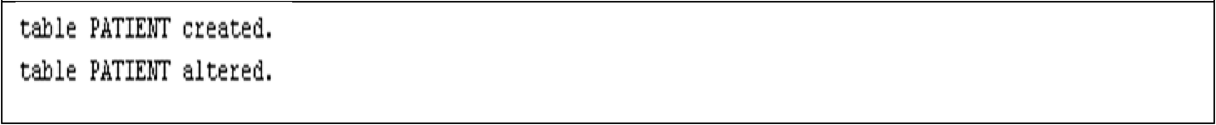


```
table PATIENT dropped.
table PATIENT created.
table PATIENT dropped.
table PATIENT created.

Error starting at line 23 in command:
INSERT INTO patient VALUES  (111,9,'John','Doe')
Error report:
SQL Error: ORA-02290: check constraint (IRAJ.PATIENT_HEIGHT_CK) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
1 rows inserted.
TABLE_NAME                     CONSTRAINT_NAME                CONSTRAINT_TYPE
------------------------------ ------------------------------ ---------------
PATIENT                        PATIENT_HEIGHT_CK              C
```

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);


--ALTER TABLE patient MODIFY height CHECK(height>10);

--Can use ADD to add a check constraint with or without a name.
--ALTER TABLE patient ADD CHECK(height>10);
--ALTER TABLE patient ADD CONSTRAINT patient_height_ck CHECK(height>10);
```

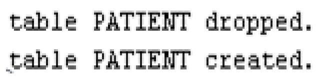


```
table PATIENT created.
table PATIENT altered.
```

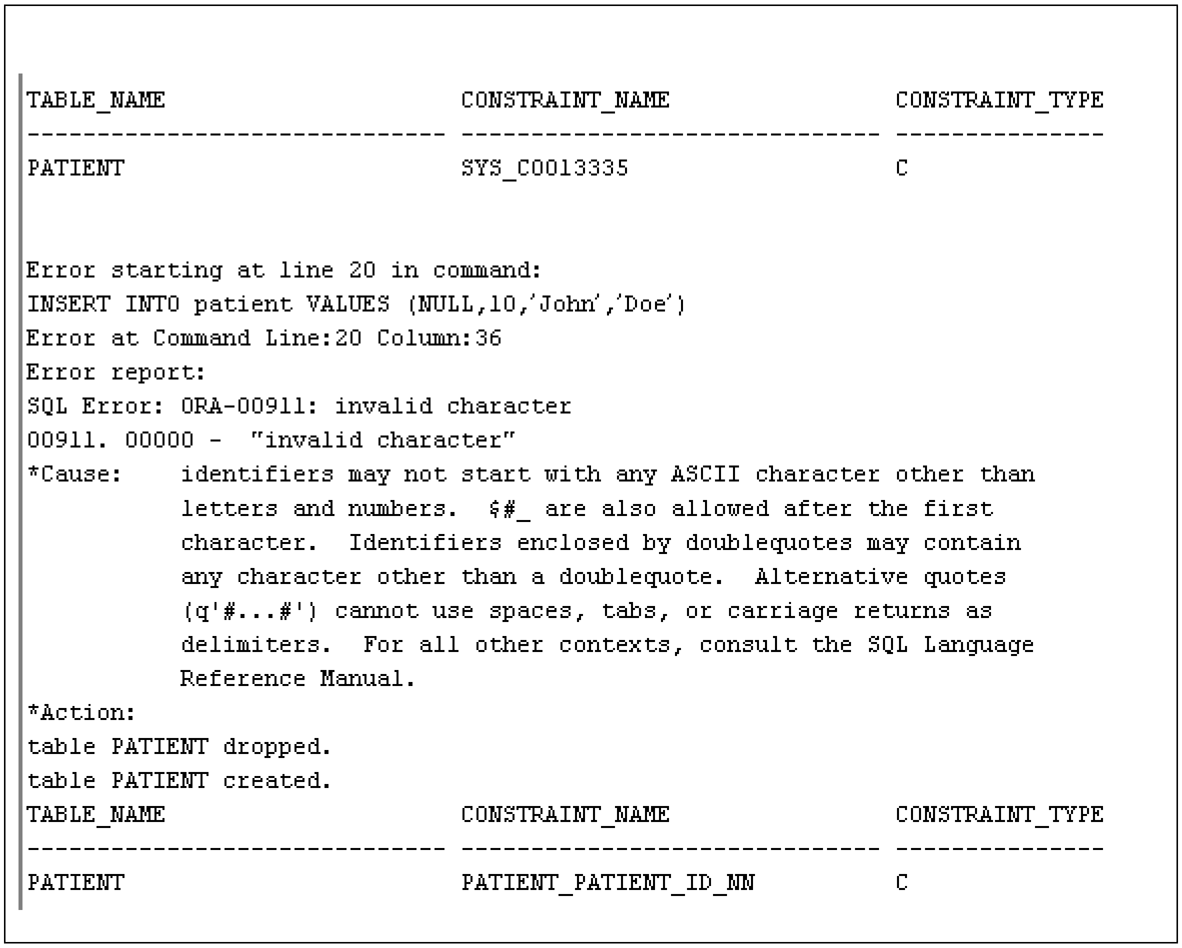
3.10 Not NULL constraint

Example 3.10a (Not NULL constraint)

```
DROP TABLE patient;





--Cannot create NULL constraint at the table level. In this example, the constraint doesn’t have a name.
--This means that data has to be provided and it can be a duplicate, unlike primary and unique keys.
--There is no such thing as a composite NOT NULL.
CREATE  TABLE Patient
(
      Patient_id  NUMBER NOT NULL,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);



--Notice the the constraint type for a not NULL constraint is (C).
SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';



--This gives an error message because of the NULL keyword. Patient id cannot be NULL
INSERT INTO patient VALUES (NULL,10,’John’,’Doe’);

DROP TABLE patient;


--Like any other constraint, you can give this constraint a name.
CREATE  TABLE Patient
(
      Patient_id  NUMBER CONSTRAINT patient_patient_id_nn NOT NULL,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);


SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';
```



```
table PATIENT dropped.
table PATIENT created.
```



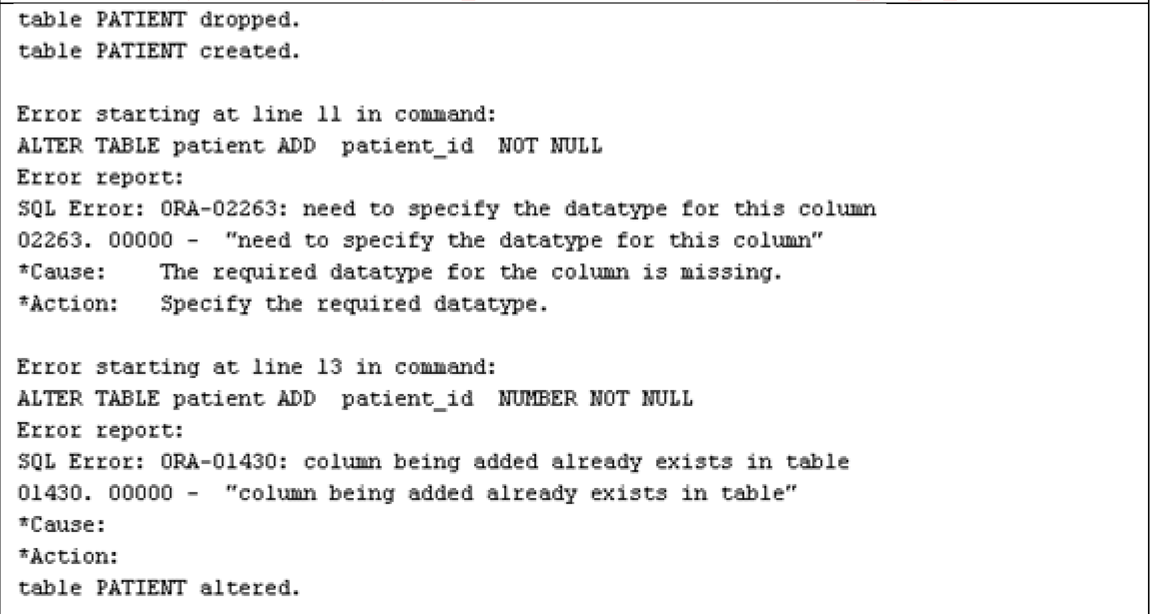
```
TABLE_NAME                     CONSTRAINT_NAME                CONSTRAINT_TYPE
------------------------------ ------------------------------ ---------------
PATIENT                        SYS_C0013335                   C


Error starting at line 20 in command:
INSERT INTO patient VALUES (NULL,10,'John','Doe')
Error at Command Line:20 Column:36
Error report:
SQL Error: ORA-00911: invalid character
00911. 00000 -  "invalid character"
*Cause:    identifiers may not start with any ASCII character other than
           letters and numbers.  $#_ are also allowed after the first
           character.  Identifiers enclosed by doublequotes may contain
           any character other than a doublequote.  Alternative quotes
           (q'#...#') cannot use spaces, tabs, or carriage returns as
           delimiters.  For all other contexts, consult the SQL Language
           Reference Manual.
*Action:
table PATIENT dropped.
table PATIENT created.
TABLE_NAME                     CONSTRAINT_NAME                CONSTRAINT_TYPE
------------------------------ ------------------------------ ---------------
PATIENT                        PATIENT_PATIENT_ID_NN          C
```

Example 3.10b (Using modify command)

Notice that only modify can be used with the ALTER command when dealing with a not NULL constraint, which may or may not include the datatype. Can only use add if the column does not exist in the table.

```
DROP TABLE patient;
CREATE  TABLE Patient
(
      Patient_id  NUMBER,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);


--Invalid: cannot add a not NULL constraint.
ALTER TABLE patient ADD  patient_id NOT NULL;


--Invalid: cannot add a not NULL constraint.
ALTER TABLE patient ADD  patient_id  NUMBER NOT NULL;
```

```
--Must use modify
ALTER TABLE patient MODIFY  patient_id  NOT NULL;


--ALTER TABLE patient MODIFY  patient_id  NUMBER NOT NULL;
--Can also give the constraint a name
--ALTER TABLE patient MODIFY  patient_id  NUMBER CONSTRAINT patient_pat_id_nn  NOT NULL;
```



```
table PATIENT dropped.
table PATIENT created.

Error starting at line 11 in command:
ALTER TABLE patient ADD  patient_id  NOT NULL
Error report:
SQL Error: ORA-02263: need to specify the datatype for this column
02263. 00000 -  "need to specify the datatype for this column"
*Cause:    The required datatype for the column is missing.
*Action:   Specify the required datatype.

Error starting at line 13 in command:
ALTER TABLE patient ADD  patient_id  NUMBER NOT NULL
Error report:
SQL Error: ORA-01430: column being added already exists in table
01430. 00000 -  "column being added already exists in table"
*Cause:
*Action:
table PATIENT altered.
```

Example 3.10c (Between clause)

More examples of check constraints. The Between clause is inclusive, which in this example means that both 10 and 100 are included.

```
DROP TABLE patient;

--An example of a different type of check constraint using the IN and BETWEEN clause.
CREATE  TABLE Patient
(
  Patient_id       NUMBER,
  Height           NUMBER CONSTRAINT patient_height_ck CHECK
                                    (height BETWEEN 5 and 10),
  gender           CHAR CHECK(gender in ('m','f')),
  Weight           NUMBER CHECK(weight BETWEEN 10 AND 100),
  Fname            VARCHAR2(20),
  Lname            VARCHAR2(20)
);
```

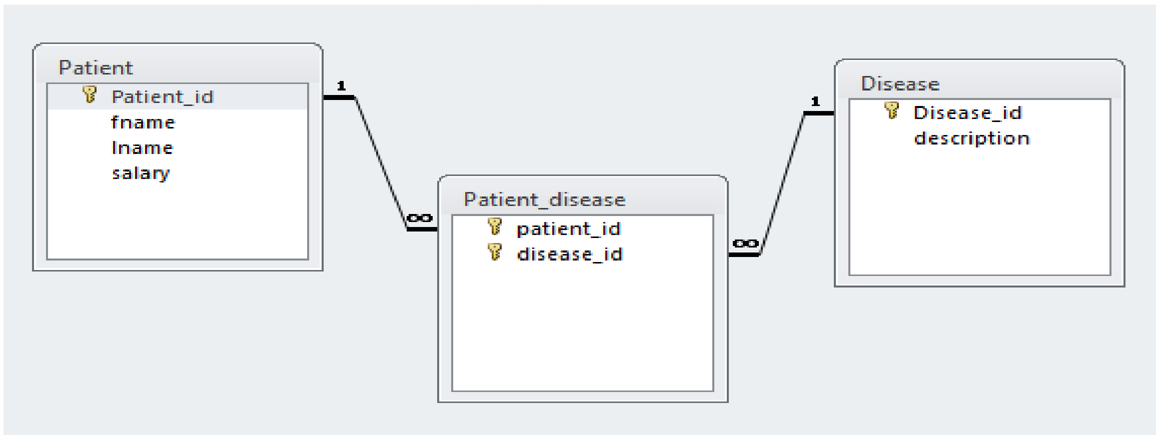
```
table PATIENT dropped.
table PATIENT created.
```

✓ *CHECK 3D*

1. Drop the PERSON table and re-create it with the primary key defined at the table level and the unique key at the column level (Person_id (PK), SSN (UK), lname, salary, DOB). Give your primary key constraint a name.
2. After the table has been created, create a NOT NULL constraint on DOB.
3. Insert data to verify that your constraints are correct.
4. Create a check constraint such that salary is always > 10000. Insert data to verify.

"Perservance is not a long race; it is many short races one after another! "

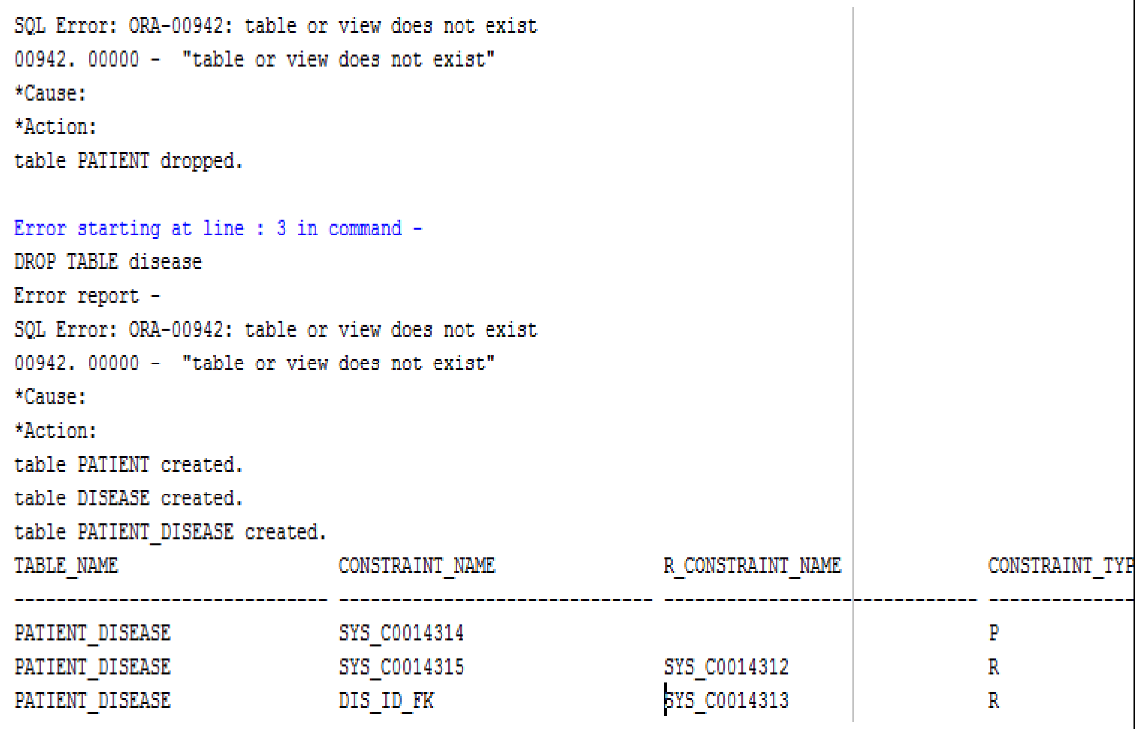
3.11 Foreign key constraint



Example 3.11a (References keyword)

The REFERENCES keyword refers to referential integrity, which means the user is referring to something that exists in another table. For example, the value entered in patient_id column of the patient_disease table references a value in the patient_id column of the patient table. The REFERENCES keyword is used to identify the table and column that must already contain the data being entered. The column referenced must be a primary key column. The child table (patient_disease) refers to the parent tables (patient, disease). The datatype of the columns that are establishing the relationship between the two tables must be the same.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;

CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height       NUMBER,
      Fname        VARCHAR2(20),
      Lname        VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid       NUMBER PRIMARY KEY,
      disease_desc    VARCHAR2(20)
);


--To create a foreign key, go to the many table and use the references keyword to point to the
--column on the one table. The column on the one side should be  a primary key.Use constraint
--keyword to give the foreign key a name.
CREATE  TABLE patient_disease
(
    Patient_id        NUMBER REFERENCES patient (patient_id),
    Disease_id        NUMBER CONSTRAINT dis_id_fk REFERENCES disease,
    PRIMARY KEY (patient_id, disease_id)
);

--The Foreign key constraint resides in the PATIENT_DISEASE table and constraint type is (R). The
--constraint name on the many side connects to the r_constraint name on the one side, which is
a
--primary key.
SELECT table_name, constraint_name,r_constraint_name, constraint_type
FROM user_constraints WHERE table_name='PATIENT_DISEASE';
```

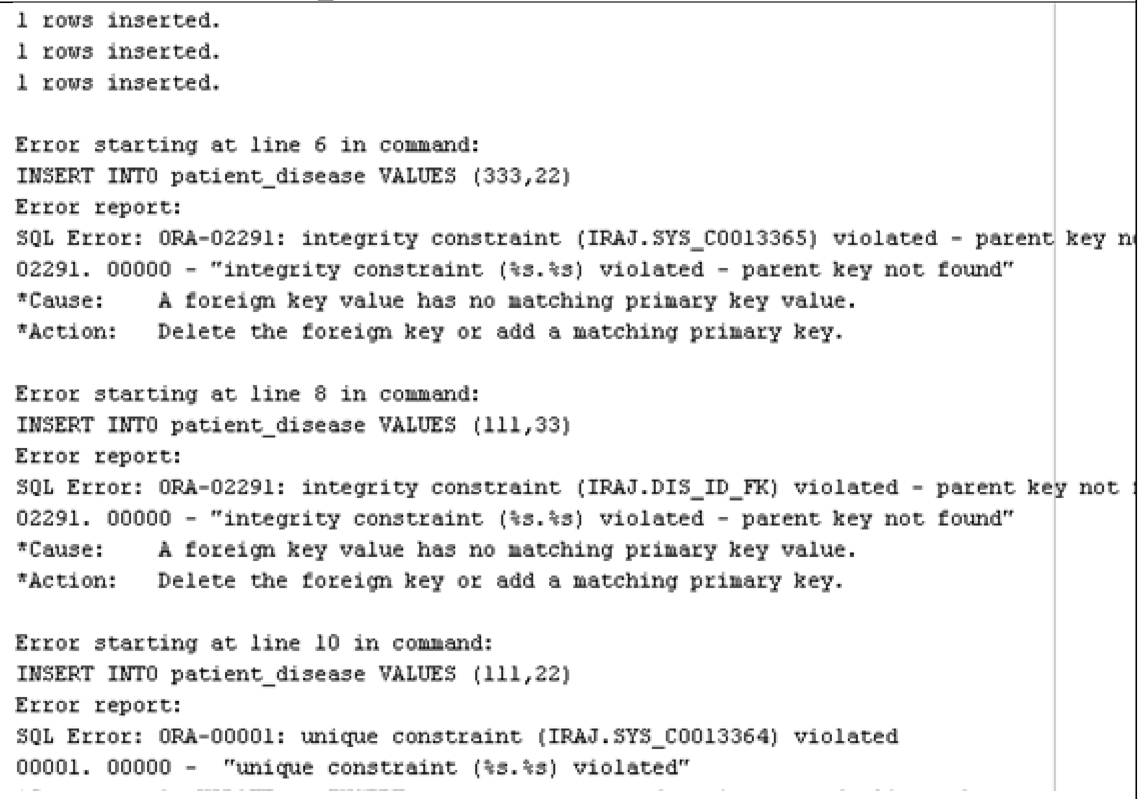


```
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
table PATIENT dropped.

Error starting at line : 3 in command -
DROP TABLE disease
Error report -
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
table PATIENT created.
table DISEASE created.
table PATIENT_DISEASE created.
TABLE_NAME                     CONSTRAINT_NAME                R_CONSTRAINT_NAME              CONSTRAINT_TYP
------------------------------ ------------------------------ ------------------------------ --------------

PATIENT_DISEASE                SYS_C0014314                                                  P
PATIENT_DISEASE                SYS_C0014315                   SYS_C0014312                   R
PATIENT_DISEASE                DIS_ID_FK                      SYS_C0014313                   R
```

Example 3.11b (Foreign keys and insert statement)

Notice that the parent tables have to be populated first before any child records are inserted. The data in the child table must match data from the parent tables.

```
INSERT INTO patient VALUES (111,90,'John','Doe');
INSERT INTO disease VALUES (22,'Malaria');



--(111) connects to the patient table whereas (22) connects to disease  table.
INSERT INTO patient_disease VALUES (111,22);

--INVALID: Connection to patient table (333) is problematic
INSERT INTO patient_disease VALUES (333,22);


--INVALID: Connection to disease table (33) is problematic
INSERT INTO patient_disease VALUES (111,33);


--INVALID: Primary key violation
INSERT INTO patient_disease VALUES (111,22);
```

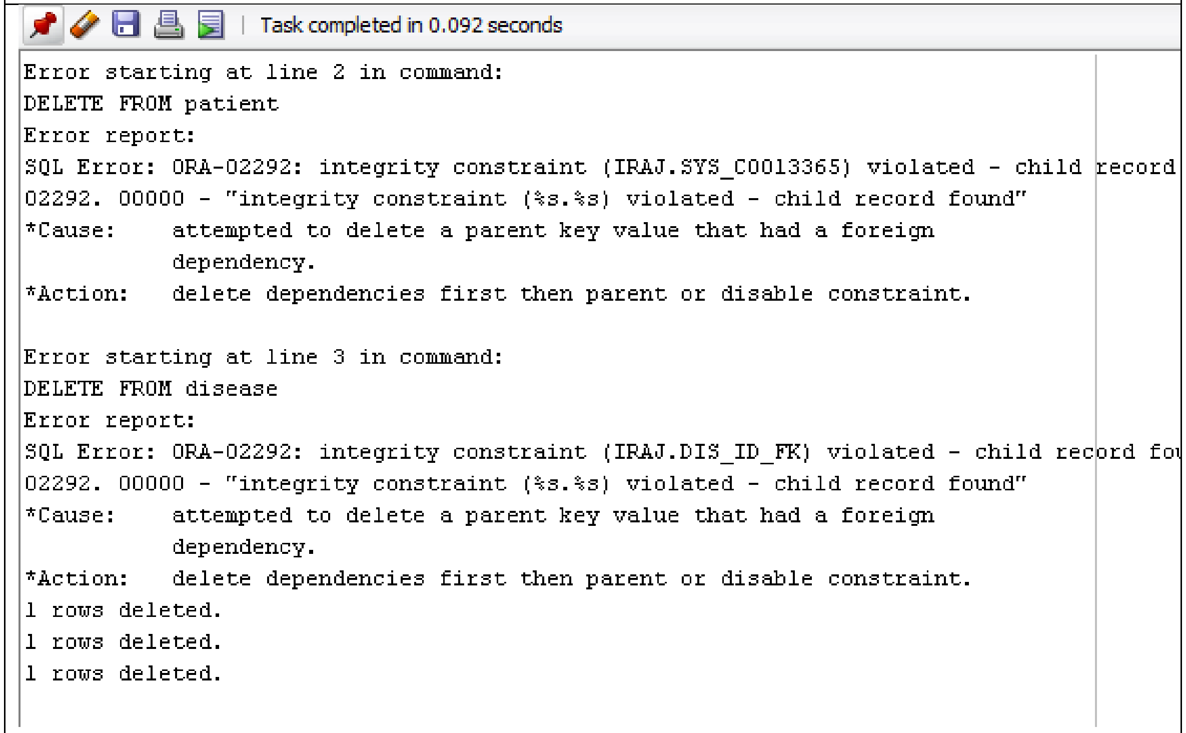


```
1 rows inserted.
1 rows inserted.
1 rows inserted.

Error starting at line 6 in command:
INSERT INTO patient_disease VALUES (333,22)
Error report:
SQL Error: ORA-02291: integrity constraint (IRAJ.SYS_C0013365) violated - parent key n
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.

Error starting at line 8 in command:
INSERT INTO patient_disease VALUES (111,33)
Error report:
SQL Error: ORA-02291: integrity constraint (IRAJ.DIS_ID_FK) violated - parent key not
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.

Error starting at line 10 in command:
INSERT INTO patient_disease VALUES (111,22)
Error report:
SQL Error: ORA-00001: unique constraint (IRAJ.SYS_C0013364) violated
00001. 00000 -  "unique constraint (%s.%s) violated"
```

Example 3.11c (Foreign keys and delete statement)

Note that the child records must be deleted prior to deleting the parent records.

```
--Both these deletes are invalid because there is a corresponding record in the patient_disease table
--that is connected to both of  these tables.
DELETE FROM patient;
DELETE FROM disease;

--Note the order of deletes. Must delete from patient_disease to get rid -of the association.
DELETE FROM patient_disease;

--The order of parent tables is not important.
DELETE FROM patient;
DELETE FROM disease;
```



Task completed in 0.092 seconds

```
Error starting at line 2 in command:
DELETE FROM patient
Error report:
SQL Error: ORA-02292: integrity constraint (IRAJ.SYS_C0013365) violated - child record
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.

Error starting at line 3 in command:
DELETE FROM disease
Error report:
SQL Error: ORA-02292: integrity constraint (IRAJ.DIS_ID_FK) violated - child record fou
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
1 rows deleted.
1 rows deleted.
1 rows deleted.
```

Example 3.11d (Truncate v. delete)

(TRUNCATE vs DELETE) Note: you cannot use TRUNCATE TABLE on a table referenced by a FOREIGN KEY constraint; instead, use DELETE statement without a WHERE clause. You have to get rid of the foreign key constraint(s) if you want to truncate the parent table.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;

CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);
```

```
CREATE  TABLE Disease
(
      diseaseid      NUMBER PRIMARY KEY,
      disease_desc   VARCHAR2(20)
);

CREATE  TABLE patient_disease
(
   Patient_id    NUMBER REFERENCES patient (patient_id),
   Disease_id        NUMBER CONSTRAINT dis_id_fk REFERENCES disease,
   PRIMARY KEY (patient_id, disease_id)
);

INSERT INTO patient VALUES (111,90,'John','Doe');
INSERT INTO disease VALUES (22,'Malaria');
INSERT INTO patient_disease VALUES (111,22);


--INVALID. Must truncate child table first.
TRUNCATE TABLE patient;


--VALID. Can only truncate child table
TRUNCATE TABLE patient_disease;


--INVALID. Unlike the delete command, parent tables cannot be truncated  unless the foreign keys
--are dropped or disabled. Use delete instead.
TRUNCATE TABLE patient;
TRUNCATE TABLE disease;
DELETE FROM patient;
DELETE FROM disease;
```

```
table PATIENT_DISEASE dropped.

table PATIENT dropped.

table DISEASE dropped.

table PATIENT created.

table DISEASE created.

table PATIENT_DISEASE created.

1 rows inserted.

1 rows inserted.

1 rows inserted.

Error starting at line : 30 in command -
```

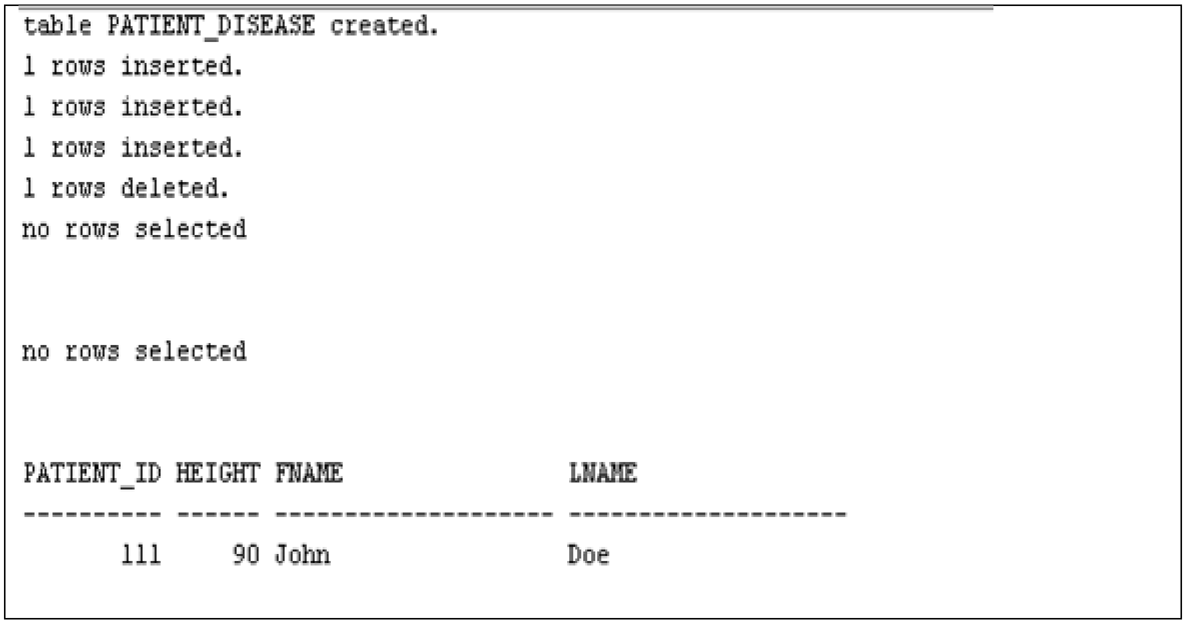
```
TRUNCATE TABLE patient
Error report -
SQL Error: ORA-02266: unique/primary keys in table referenced by enabled foreign keys
02266. 00000 -  "unique/primary keys in table referenced by enabled foreign keys"
*Cause:    An attempt was made to truncate a table with unique or
           primary keys referenced by foreign keys enabled in another table.
           Other operations not allowed are dropping/truncating a partition of a
           partitioned table or an ALTER TABLE EXCHANGE PARTITION.
*Action:   Before performing the above operations the table, disable the
           foreign key constraints in other tables. You can see what
           constraints are referencing a table by issuing the following
           command:
           SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";
table PATIENT_DISEASE truncated.
Error starting at line : 37 in command -
TRUNCATE TABLE patient
Error report -
SQL Error: ORA-02266: unique/primary keys in table referenced by enabled foreign keys
02266. 00000 -  "unique/primary keys in table referenced by enabled foreign keys"
*Cause:    An attempt was made to truncate a table with unique or
           primary keys referenced by foreign keys enabled in another table.
           Other operations not allowed are dropping/truncating a partition of a
           partitioned table or an ALTER TABLE EXCHANGE PARTITION.
*Action:   Before performing the above operations the table, disable the
           foreign key constraints in other tables. You can see what
```

```
          constraints are referencing a table by issuing the following
          command:
          SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";


Error starting at line : 38 in command -
TRUNCATE TABLE disease
Error report -
SQL Error: ORA-02266: unique/primary keys in table referenced by enabled foreign keys
02266. 00000 -  "unique/primary keys in table referenced by enabled foreign keys"
*Cause:    An attempt was made to truncate a table with unique or
          primary keys referenced by foreign keys enabled in another table.
          Other operations not allowed are dropping/truncating a partition of a
          partitioned table or an ALTER TABLE EXCHANGE PARTITION.
*Action:   Before performing the above operations the table, disable the
          foreign key constraints in other tables. You can see what
          constraints are referencing a table by issuing the following
          command:
          SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";
1 rows deleted.
1 rows deleted.
```

Example 3.11e (Delete cascade)

Delete cascade option allows for a record in the parent table to be deleted even if it is being referenced in the child table. Using this option will remove all the corresponding records in the child table in order to maintain referential integrity.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;

CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid      NUMBER PRIMARY KEY,
      disease_desc   VARCHAR2(20)
);

CREATE  TABLE patient_disease
(
   Patient_id        NUMBER REFERENCES patient ON DELETE CASCADE,
   Disease_id        NUMBER REFERENCES disease ON DELETE CASCADE,
   PRIMARY KEY (patient_id, disease_id)
);

--Notice that first the parent tables are populated and then the child tabls. Also notice how the
--foreign keys match up with the primary keys.
INSERT INTO patient VALUES (111,90,'John','Doe');
INSERT INTO disease VALUES (22,'Malaria');
INSERT INTO patient_disease VALUES (111,22);

/* This would usually fail because there are child records that are connected to this table and so the
children record have to be deleted first. However, with cascade delete, when a record is deleted
from the parent table, all the related records in the child table are automatically deleted as well. */
DELETE FROM disease;
SELECT * FROM disease;
SELECT * FROM patient_disease;
SELECT * FROM patient;
```

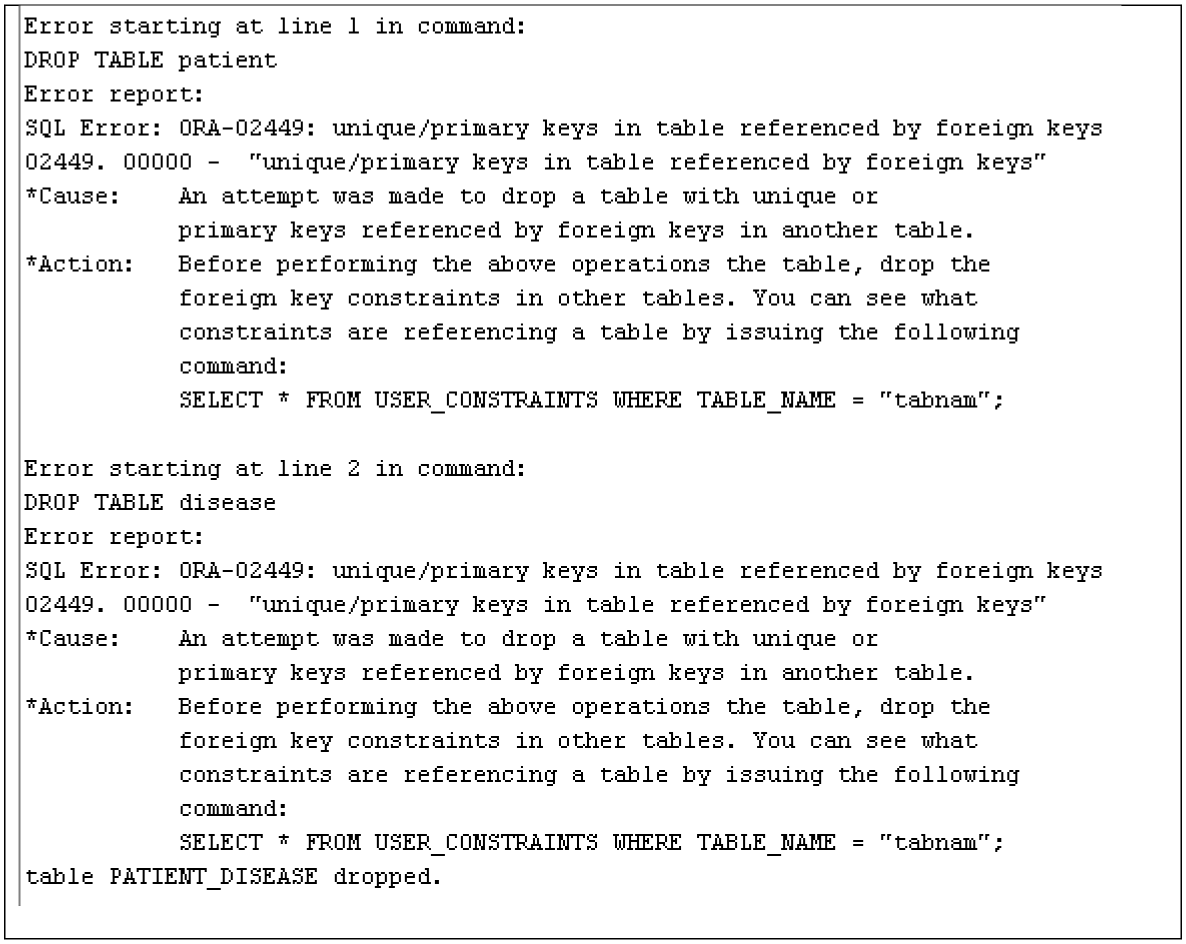


```
table PATIENT_DISEASE created.
1 rows inserted.
1 rows inserted.
1 rows inserted.
1 rows deleted.
no rows selected


no rows selected


PATIENT_ID HEIGHT FNAME                LNAME
---------- ------ -------------------- --------------------
       111     90 John                 Doe
```

Example 3.11f (Foreign keys and dropping tables)

When getting rid of the tables, the child table has to be deleted prior to the parents.

```
--The child table must be dropped first and then the parent tables can be dropped in any order.
--This will cause an error.
DROP TABLE patient;
DROP TABLE disease;
DROP TABLE patient_disease;
```



```
Error starting at line 1 in command:
DROP TABLE patient
Error report:
SQL Error: ORA-02449: unique/primary keys in table referenced by foreign keys
02449. 00000 -  "unique/primary keys in table referenced by foreign keys"
*Cause:    An attempt was made to drop a table with unique or
           primary keys referenced by foreign keys in another table.
*Action:   Before performing the above operations the table, drop the
           foreign key constraints in other tables. You can see what
           constraints are referencing a table by issuing the following
           command:
           SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";

Error starting at line 2 in command:
DROP TABLE disease
Error report:
SQL Error: ORA-02449: unique/primary keys in table referenced by foreign keys
02449. 00000 -  "unique/primary keys in table referenced by foreign keys"
*Cause:    An attempt was made to drop a table with unique or
           primary keys referenced by foreign keys in another table.
*Action:   Before performing the above operations the table, drop the
           foreign key constraints in other tables. You can see what
           constraints are referencing a table by issuing the following
           command:
           SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = "tabnam";
table PATIENT_DISEASE dropped.
```

Example 3.11g (Foreign key at the table level)

Notice the syntax FOREIGN KEY is used to refer to the column in the child table and the references syntax is used to refer to the column in the parent table.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;

CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid      NUMBER PRIMARY KEY,
      disease_desc   VARCHAR2(20)
);
```

```
--Foreign key is being created at the table level. This requires the additional -(FOREIGN KEY) keyword.
--Since all the columns have been defined in the child table already, we need to identify the
--column that will be the foreign key in the child table and then we use the REFERENECES keyword
--as in the previous syntax. Once again the CONSTRAINT keyword is needed to give it a name.
CREATE  TABLE patient_disease
(
   Patient_id        NUMBER,
   Disease_id        NUMBER,
   PRIMARY KEY (patient_id, disease_id),
   FOREIGN KEY (patient_id) REFERENCES patient ON DELETE CASCADE,
   CONSTRAINT patient_disease_disease_id_fk FOREIGN KEY (disease_id)
   REFERENCES disease
);
```

Example 3.11h (Using alter table to create foreign keys)

Notice the syntax FOREIGN KEY is used to refer to the column in the child table and the references syntax is used to refer to the column in the parent table.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;

CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid     NUMBER PRIMARY KEY,
      disease_desc  VARCHAR2(20)
);


CREATE  TABLE patient_disease
(
      Patient_id     NUMBER,
      Disease_id     NUMBER,
      PRIMARY KEY     (patient_id, disease_id)
);

--Alter table is used to identify a foreign key
ALTER TABLE patient_disease ADD FOREIGN KEY(patient_id) REFERENCES
patient;

ALTER TABLE patient_disease ADD CONSTRAINT
patient_disease_disease_id_fk FOREIGN KEY(disease_id) REFERENCES
disease;
```

```
Error starting at line : 1 in command -
DROP TABLE patient_disease
Error report -
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
table PATIENT dropped.
Error starting at line : 3 in command -
DROP TABLE disease
Error report -
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
table PATIENT created.
table DISEASE created.
table PATIENT_DISEASE created.
table PATIENT_DISEASE altered.
table PATIENT_DISEASE altered.
```

✓ *CHECK 3E*

1. Given the tables Person and Persnality, create the foreign key relationship:
    a. At the table level (With a constraint name)
    b. At the column level (Without a constraint name)
    c. After the tables have been created
2. Insert to verify.
3. Drop foreign key constraint and re-create with DELETE CASCADE option.
4. Insert and delete to verify constraint.
5. Truncate both tables.

"My father always used to say that when you die, if you've got five real friends, then you've had a great life. "

3.12 Disabling/Enabling/Dropping constraints

To DISABLE a constraint, you issue an ALTER TABLE command and change the constraint's status to DISABLE. Later, you can reissue the ALTER TABLE command and change the constraint's status back to ENABLE.After the constraint is enabled, data added or modified is again checked by the constraint.

Example 3.12a (Disable/Enable constraints)

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;
CREATE  TABLE Patient
(
      Patient_id  NUMBER CONSTRAINT patient_patient_id_pk PRIMARY
KEY,
      Height      NUMBER CONSTRAINT patient_height_ck CHECK
(height>4),
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      CONSTRAINT patient_fname_lname_uk UNIQUE(fname,lname)
);

--By disabling the constraint, records can be inserted which would have created a problem with
the
--constraint. A disabled constraint can  later be enabled. The primary key can be enabled or
--disabled using just the PRIMARY KEY syntax with or without the columns.
ALTER TABLE patient DISABLE PRIMARY KEY;

--The status column conveys if the constraint is enabled or disabled
SELECT constraint_name, constraint_type, status FROM user_constraints
WHERE table_name='PATIENT';
```

```
table PATIENT created.

table PATIENT altered.


CONSTRAINT_NAME             CONSTRAINT_TYPE       STATUS

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

PATIENT_HEIGHT_CK             C                   ENABLED

PATIENT_PATIENT_ID_PK         P                   DISABLED
```

```
PATIENT_FNAME_LNAME_UK   U                    ENABLED
```

```
ALTER TABLE patient ENABLE PRIMARY KEY;
ALTER TABLE patient DISABLE CONSTRAINT patient_patient_id_pk;
ALTER TABLE patient DISABLE CONSTRAINT patient_fname_lname_uk;
ALTER TABLE patient ENABLE CONSTRAINT patient_fname_lname_uk;
ALTER TABLE patient DISABLE UNIQUE (fname,lname);
```

```
table PATIENT altered.

table PATIENT altered.

table PATIENT altered.

table PATIENT altered.

table PATIENT altered.
```

If you create a constraint, you can delete it from the table with the DROP (constraintname) command. In addition, if you need to change or modify a constraint, your only option is to delete the constraint and then create a new one. You use the ALTER TABLE command to drop an existing constraint from a table. If this ALTER TABLE command is executed successfully, the constraint no longer exists, and any value is accepted as input to the column.

- The DROP clause varies depending on the type of constraint being deleted. If the DROP clause references the PRIMARY KEY constraint for the table, using the keywords PRIMARY KEY is enough because only one such clause is allowed for each table in the database.
- To delete a UNIQUE constraint, only the column name affected by the constraint is required because a column is referenced by only one UNIQUE constraint.
- Any other type of constraint must be referenced by the constraint's actual name—regardless of whether the constraint name is assigned by a user or the Oracle server.

Example 3.12b (Drop constraints)

```
DROP TABLE patient;


CREATE  TABLE Patient
(
      Patient_id  NUMBER CONSTRAINT patient_patient_id_pk PRIMARY KEY,
      Height      NUMBER CONSTRAINT patient_height_ck CHECK (height>4),
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      CONSTRAINT  patient_fname_lname_uk UNIQUE(fname,lname)
);


--Constraints can also be dropped. Once dropped, there is no way to get them back. They have to be
--recreated, which is different from disabled constraints.
ALTER TABLE patient DROP CONSTRAINT patient_height_ck;
ALTER TABLE patient DROP PRIMARY KEY;
ALTER TABLE patient DROP UNIQUE (fname,lname);
```

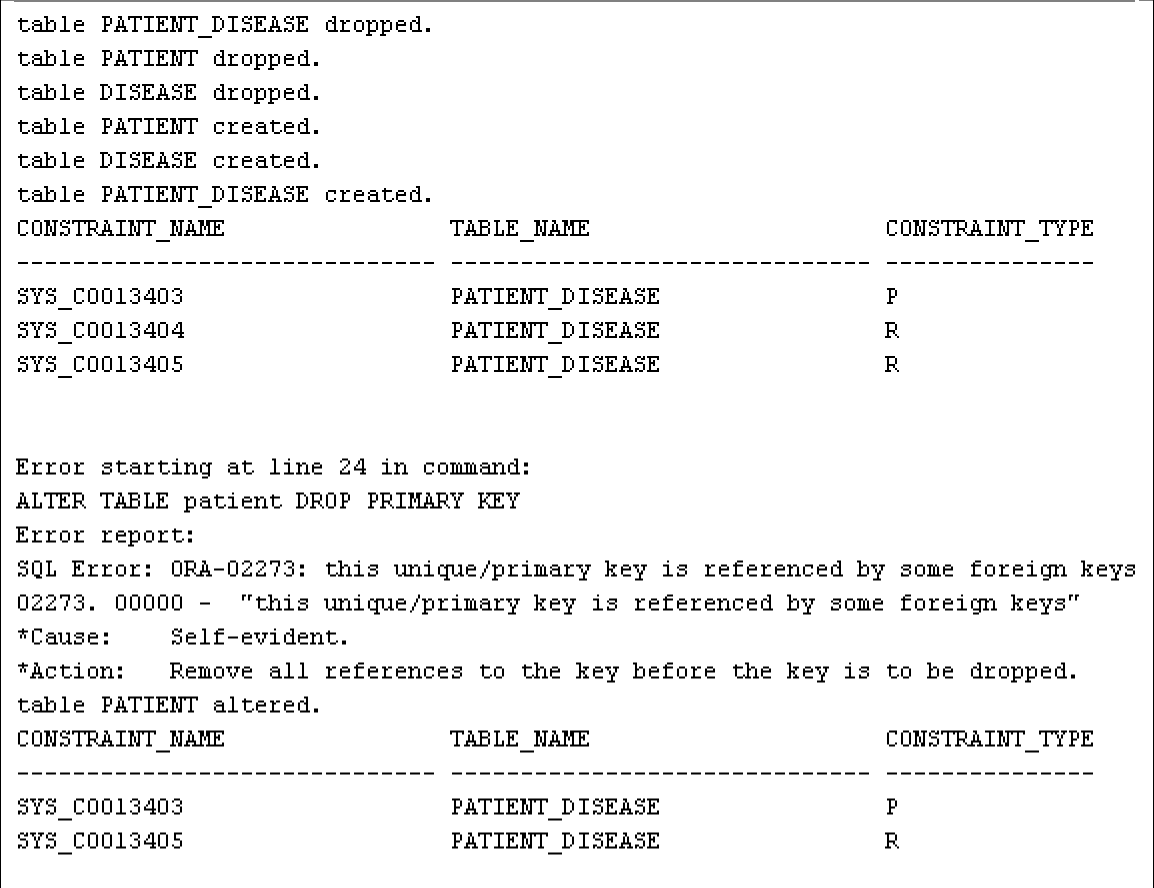
```
table PATIENT dropped.
table PATIENT created.
table PATIENT altered.
table PATIENT altered.
table PATIENT altered.
```

Example 3.12c (Primary key with a cascade option)

If this ALTER TABLE command is executed successfully, the constraint no longer exists, and any value is accepted as input to the column. If needed, the associated FOREIGN KEY can be deleted along with the PRIMARY KEY deletion by using the CASCADE option.

```
DROP TABLE patient_disease;
DROP TABLE patient;
DROP TABLE disease;


CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20)
);

CREATE  TABLE Disease
(
      diseaseid      NUMBER PRIMARY KEY,
      disease_desc   VARCHAR2(20)
);

CREATE  TABLE patient_disease
(
       Patient_id    NUMBER REFERENCES patient (patient_id),
       Disease_id    NUMBER REFERENCES disease,
       PRIMARY KEY (patient_id, disease_id)
);
```

```
--Notice there are two foreign key.s
SELECT constraint_name, table_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT_DISEASE';

--This is going to cause a problem because a foreign key is attached to it.
ALTER TABLE patient DROP PRIMARY KEY;

--This will automatically get rid of the foreign key constraint that  is attached to this primary key.
ALTER TABLE patient DROP PRIMARY KEY CASCADE;

--Notice, there is only one foreign key.
SELECT constraint_name, table_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT_DISEASE';
```



```
table PATIENT_DISEASE dropped.
table PATIENT dropped.
table DISEASE dropped.
table PATIENT created.
table DISEASE created.
table PATIENT_DISEASE created.
CONSTRAINT_NAME                TABLE_NAME                     CONSTRAINT_TYPE
------------------------------ ------------------------------ ---------------
SYS_C0013403                   PATIENT_DISEASE                P
SYS_C0013404                   PATIENT_DISEASE                R
SYS_C0013405                   PATIENT_DISEASE                R


Error starting at line 24 in command:
ALTER TABLE patient DROP PRIMARY KEY
Error report:
SQL Error: ORA-02273: this unique/primary key is referenced by some foreign keys
02273. 00000 -  "this unique/primary key is referenced by some foreign keys"
*Cause:    Self-evident.
*Action:   Remove all references to the key before the key is to be dropped.
table PATIENT altered.
CONSTRAINT_NAME                TABLE_NAME                     CONSTRAINT_TYPE
------------------------------ ------------------------------ ---------------
SYS_C0013403                   PATIENT_DISEASE                P
SYS_C0013405                   PATIENT_DISEASE                R
```

✓ *CHECK 3F*

1. Drop the primary key constraint in the Person table.
2. Disable the unique key in the Person table.
3. Insert a record to verify.

"The real measure of our wealth is how much we'd be worth if we lost all our money"

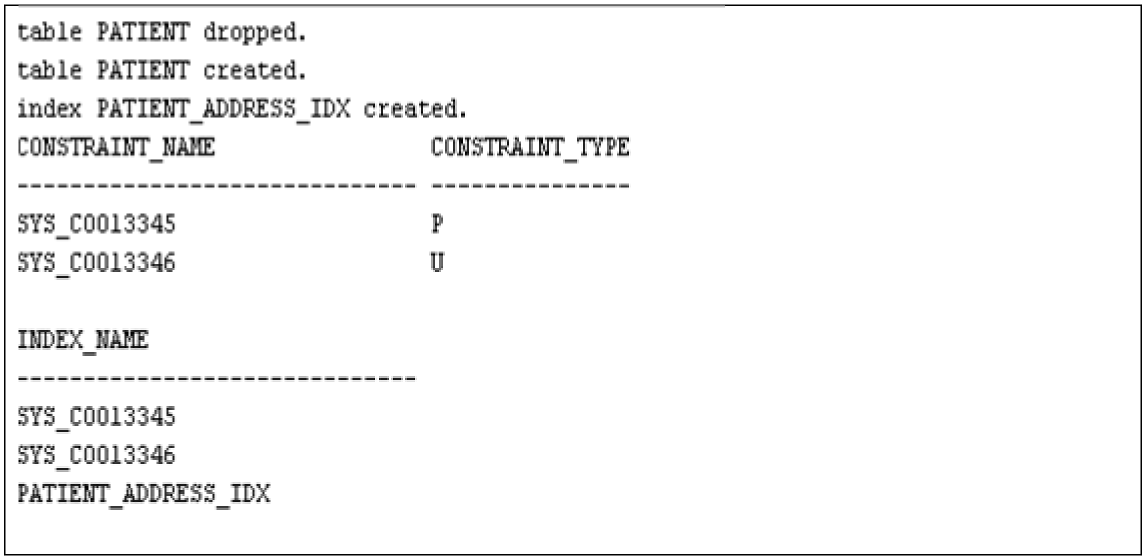
3.13 Indexes

A database index is much like the index at the end of a book. As rows of data are inserted, they’re physically added to the table in no particular order. As rows are deleted, the space can be reused by new rows. Therefore, if a search condition such as “ WHERE zip = 90404” is included in a SELECT statement, a full table scan is performed. In a full table scan, each row of the table is read, and the zip value is checked to determine whether it satisfies the condition. This issue can be addressed by applying indexes to table columns. The B- tree (balanced- tree) index is the most common index used in Oracle.

Example 3.13a (Indexes)

An index can be created implicitly or explicitly. Oracle creates an index automatically when a PRIMARY KEY or UNIQUE constraint is created for a column.

```
DROP TABLE patient;
--Indxes will be created automatically with primary keys and unique keys.
CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Address     VARCHAR2(30),
      Height      NUMBER,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      UNIQUE (fname,lname)
);

--In addition, indexes can be created manually.
CREATE INDEX patient_address_idx ON patient (address);

--The primary key and unique key information will be in the user_constraints table.
SELECT constraint_name, constraint_type FROM user_constraints WHERE
table_name='PATIENT';

--The index that was created using create index command along with the primary key and unique
--key indexes will be in this table. The name of the indexes for the primary key and unique will be
--the same as the constraint names that appear in the user_constraints table
SELECT index_name FROM user_indexes WHERE table_name='PATIENT';
```



```
table PATIENT dropped.
table PATIENT created.
index PATIENT_ADDRESS_IDX created.
CONSTRAINT_NAME                CONSTRAINT_TYPE
------------------------------ ---------------
SYS_C0013345                   P
SYS_C0013346                   U

INDEX_NAME
------------------------------
SYS_C0013345
SYS_C0013346
PATIENT_ADDRESS_IDX
```

Example 3.13b (Alter index command)

The only modification you can perform on an existing index is a name change. If you need to change the name of an index, use the ALTER INDEX command.

```
--The name of the index is altered.
ALTER INDEX patient_address_idx RENAME TO pat_add_idx;
```

Example 3.13c (Dropping an index)

DO NOT USE the ALTER command for dropping indexes. It does not work.

```
--To drop an index, use DROP INDEX.  Many make the common mistake of using the ALTER TABLE
--command but that would be wrong. Also, when a table is dropped, all the associated  constraints
--and indexes are dropped as well.
DROP INDEX patient_address_idx;
```

✓ CHECK 3G

1) When are indexes created automatically?
2) Create an index on the Person table based on lname and salary.
3) Drop the index.

"If we could read the secret history of our enemies, we should find in each person's life sorrow and suffering enough to disarm all hostility. "

Summary example

```
DROP TABLE patient_disease;
DROP TABLE disease;
DROP TABLE patient;
DROP TABLE sickperson;

--Notice that primary key is two words.
--Use the constraint keyword to name a constraint.
--Use the word unique, not unique key to create a unique constraint.
--In this example, all constraints with the exception of unique are at the column level
--NOT NULL, NULL and DEFAULT must be defined at the column level
--Composite keys must be defined at the table level.
 --Can have many unique keys but only one primary key.
CREATE  TABLE patient
(
   patient_id     NUMBER PRIMARY KEY,
   height         NUMBER CHECK (height BETWEEN 10 AND 20),
   fname          VARCHAR2(20) CONSTRAINT patient_fname_nn NOT NULL,
   lname          VARCHAR2(20),
   salary         NUMBER    DEFAULT 10000,
   CONSTRAINT patient_uk UNIQUE(lname) ,
   UNIQUE(height,salary)
);

CREATE  TABLE disease
(
      diseaseid      NUMBER PRIMARY KEY,
      disease_desc   VARCHAR2(20)
);

--The parent tables can be created in any order.
--Foreign keys can be created at the table level or column level. When creating foreign keys at
--table level, use FOREIGN KEY syntax.
--Composite keys have to be created at the table level.
--ON DELETE CASCADE allows the deletion of parent records which will automatically
--trigger the deletion of the associated child records.
CREATE  TABLE patient_disease
(
   patient_id        NUMBER REFERENCES patient ON DELETE CASCADE,
   disease_id        NUMBER,
   CONSTRAINT patient_disease_dis_id_fk FOREIGN KEY(disease_id)
   REFERENCES disease (diseaseid),
   PRIMARY KEY (patient_id, disease_id)
);

--With the foreign key in place, parent records must be inserted before the child records;
--otherwise there will be an integrity error.
INSERT INTO disease VALUES (22,'malaria');
INSERT INTO patient VALUES (111,15,'john','james');
INSERT INTO patient_disease VALUES (111,22);
SELECT * FROM patient;
```

```
--Additional columns can be added.
ALTER TABLE patient ADD DOB DATE UNIQUE;

--Columns can be renamed.
ALTER TABLE patient RENAME COLUMN DOB TO dateofbirth;

--A constraint that is disabled can be also enabled (ENABLE). In this case, because it is attached
--to a foreign key, it cannot be disabled. The foreign key will have to be removed first.
ALTER TABLE patient DISABLE PRIMARY KEY;

--Drops the column height and all its data.
ALTER TABLE patient DROP (height);

--Drops the constraint.
ALTER TABLE patient DROP CONSTRAINT patient_fname_nn;

--Constraint information for the patient table.
SELECT table_name, constraint_name, constraint_type FROM
user_constraints WHERE table_name='PATIENT';

 --This can only be done with the cascade delete option. Without this option the  children
--records will have to be deleted first.
DELETE FROM patient;

--All references to the patient table are changed to sickperson .
RENAME patient TO sickperson;
DESC sickperson;

--Deletes all the records from the table sickperson (can get it back through rollback).
DELETE FROM sickperson;

 --Deletes all th records from the table sickperson but you cannot get it back.
TRUNCATE TABLE sickperson;

--Create an index on fname. An index is automatically created with unique and primary keys.
CREATE INDEX myname ON sickperson (fname);

--Drop the index.
DROP INDEX myname;

--Some basic constraint information for tables stored in the system table user_constraints.
SELECT table_name, constraint_name, constraint_type,
search_condition FROM user_constraints WHERE table_name='NAME OF
YOUR TABLE IN UPPER CASE';

 --The table, the constraints and all the records in the table are eliminated.
--If there is a parent table, then the child table must first be dropped.
DROP TABLE myname;
```