

The Relational Data Model and Relational Database Constraints

Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
 - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.
- The first commercial implementations of the relational model became available in the early 1980s, such as the SQL/DS system on the MVS operating system by IBM and the Oracle DBMS.
- Current popular relational DBMSs (RDBMSs) include DB2 and Informix Dynamic Server (IBM), Oracle and Rdb (Oracle), Sybase DBMS (Sybase) and SQLServer and Access (Microsoft).

Relational Model Concepts

- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.

- In addition, several open source RDMSs, such as MySQL and PostgreSQL, are available.

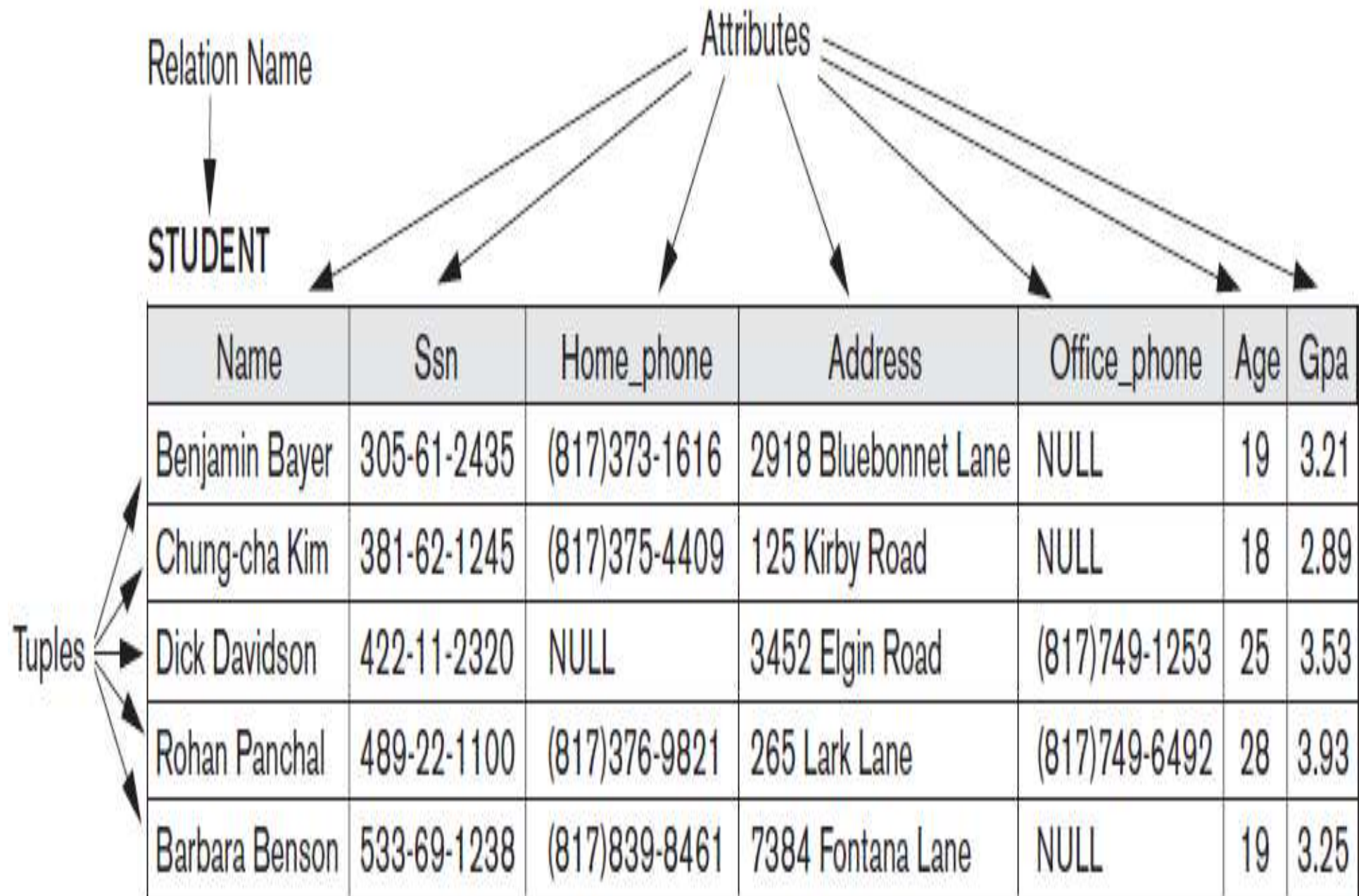
INFORMAL DEFINITIONS

- The relational model represents the database as a collection of *relations*.
- *Informally*, each relation resembles a table of values or, to some extent, a *flat file of records* (because each record has a simple linear or *flat structure*).
- When a relation is thought of as a **table of values**, **each row in the table represents** a collection of related data values.
- A row represents a fact that typically corresponds to a real-world entity or relationship.
- The table name and column names are used to help to interpret the meaning of the values in each row.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

- All values in a column are of the same data type.
- In the formal relational model terminology, a row is called a **tuple**, a column header is called an **attribute**, and the table is called a **relation**.
- The data type describing the types of values that can appear in each column is represented by a domain of possible values.



FORMAL DEFINITIONS

- A **domain**, D is a set of atomic (indivisible)values.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- Domain should have a name, which helps in interpreting its values.

Eg: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.

- A domain may have a data-type or a format defined for it.

Eg:The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit.

E.g., Dates have various formats such as mm-dd-yyyy or yyyy-mm-dd, or dd mm,yyyy etc.

- A domain is thus given a name, data type, and format.
- A **relation schema** R, denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes, A_1, A_2, \dots, A_n .
Eg: CUSTOMER (Cust-id, Cust-name, Address, Phone#)
- Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values.

- Each attribute A_i is the name of a role played by some domain D in the relation schema R .
- D is called the domain of A_i and is denoted by $\text{dom}(A_i)$.
- A relation schema is used to describe a relation; R is called the name of this relation.
- The degree (or arity) of a relation is the number of attributes n of its relation schema.

- A **tuple** is an ordered set of values
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.

Eg: <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">

is a tuple belonging to the CUSTOMER relation.

- A relation may be regarded as a *set of tuples* (rows).
- Columns in a table are also called attributes of the relation.

- A relation of degree 7 would contain seven attributes:

Eg: STUDENT(Name, Ssn, Home_phone, Address, Office_phone,
Age, Gpa)

- The relation is formed over the cartesian product of the sets;
 - each set has values from a domain;
 - that domain is used in a specific role which is conveyed by the attribute name.

Eg: attribute Cust-name is defined over the domain of strings of 25 characters.

The role these strings play in the CUSTOMER relation is that of the name of customers.

- A relation (or relation state) $r(R)$ is a *mathematical relation* of degree n on the domains $dom(A1)$, $dom(A2)$, ..., $dom(An)$, which is a subset of the cartesian product of the domains that define R :

$$r(R) \subseteq (dom(A1) \times dom(A2) \times \dots \times dom(An))$$

- R : schema of the relation is also called the **intension** of a relation
- $r(R)$: a specific "value" or population of R is also called the **extension** of a relation

- Let $S1 = \{0,1\}$
- Let $S2 = \{a,b,c\}$
- Let $R \subset S1 \times S2$
- Then for example: $r(R) = \{ \langle 0,a \rangle , \langle 0,b \rangle , \langle 1,c \rangle \}$
is one possible “state” or “population” or
“extension” r of the relation R , defined over domains
 $S1$ and $S2$. It has three tuples.

DEFINITION SUMMARY

Informal Terms

Table

Column

Row

Values in a column

Table Definition

Populated Table

Formal Terms

Relation

Attribute/Domain

Tuple

Domain

Schema of a Relation

Extension

CHARACTERISTICS OF RELATIONS

1. **Ordering of tuples in a relation $r(R)$:** The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
 - A relation is not sensitive to the ordering of tuples
 - Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level.
 - Many tuple orders can be specified on the same relation.
2. **Ordering of attributes in a relation schema R (and of values within each tuple):**
 - As per the definition of a relation, an n -tuple is an ordered list of n values, so the ordering of values in a tuple—and hence of attributes in a relation schema—is important.
 - We will consider the attributes in $R(A_1, A_2, \dots, A_n)$ and the values in $t = \langle v_1, v_2, \dots, v_n \rangle$ to be *ordered*.

3. Values and NULLs in a tuple: All tuple values are considered *atomic* (indivisible).

- Hence, composite and multivalued attributes are not allowed. This model is called the **flat relational model**. Theory behind the relational model was developed with **first normal form assumption**.
- Hence, multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes in the basic relational model.
- A special null value is used to represent values, such as *value unknown*, *value exists but is not available*, or *attribute does not apply to this tuple* (also known as *value undefined*).

- During database design, it is best to avoid NULL values as much as possible.

4. Interpretation (Meaning) of a Relation: The relation schema can be interpreted as a **declaration or a type of assertion**.

Eg: the schema of the STUDENT relation asserts that, in general, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa.

- Each tuple in the relation can then be interpreted as a **fact or a particular instance** of the assertion.

Eg: the first tuple in STUDENT relation asserts the fact that there is a STUDENT whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on.

Relational Model Notation

- A relation schema R of degree n is denoted by $R(A_1, A_2, \dots, A_n)$.
- The uppercase letters Q, R, S denote relation names.
- The lowercase q, r, s denote relation states.
- The letters t, u, v denote tuples.
- In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation *whereas* STUDENT(Name, Ssn, ...) refers *only to the relation schema*.
- An attribute A can be qualified with the relation name R to which it belongs by using the dot notation $R.A$

Eg: STUDENT.Name

Reason: The same name may be used for two attributes in different relations. However, all attribute names *in a particular relation* must be distinct.

- An n -tuple t in a relation $r(R)$ is denoted by $t = \langle v_1, v_2, \dots, v_n \rangle$, where v_i is the value corresponding to attribute A_i .
- The following notation refers to **component values** of tuples:
 - Both $t[A_i]$ and $t.A_i$ (and sometimes $t[i]$) refer to the value v_i in t for attribute A_i .
 - Both $t[A_u, A_w, \dots, A_z]$ and $t.(A_u, A_w, \dots, A_z)$, where A_u, A_w, \dots, A_z is a list of attributes from R , refer to the subtuple of values $\langle v_u, v_w, \dots, v_z \rangle$ from t corresponding to the attributes specified in the list.

Eg: tuple $t = \langle \text{'Barbara Benson'}, \text{'533-69-1238'}, \text{'(817)839-8461'}, \text{'7384 Fontana Lane'}, \text{NULL}, 19, 3.25 \rangle$ from the STUDENT relation we have $t[\text{Name}] = \langle \text{'Barbara Benson'} \rangle$, and $t[\text{Ssn}, \text{Gpa}, \text{Age}] = \langle \text{'533-69-1238'}, 3.25, 19 \rangle$.

Relational Model Constraints and Relational Database Schemas

- In a relational database, there will be many relations, and the tuples in those relations are usually **related** in various ways.
- The **state of the whole database** will correspond to the states of all its relations at a particular point in time.
- There are generally many **restrictions or constraints** (derived from the rules in the mini-world that the database represents) on the actual values in a database state.
- Various restrictions on data in the mini world can be specified on a relational database in the form of constraints.
- Constraints are *conditions* that must hold on *all* valid relation instances.

● Constraints on databases can generally be divided into three main categories:

1. Constraints that are inherent in the data model are called inherent model-based or implicit constraints.

Eg: A relation cannot have duplicate tuples.

2. Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL are schema-based constraints or explicit constraints.

3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs are application-based or semantic constraints or business rules.

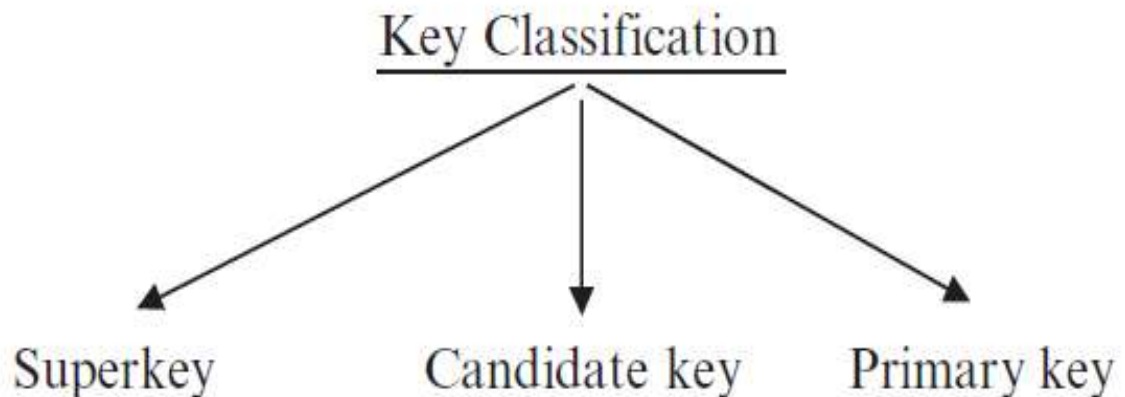
● The type of constraints that can be expressed in the relational model are the schema-based constraints which include :

1. Domain constraints
2. Key constraints
3. Constraints on NULLs
4. Entity integrity constraints
5. Referential integrity constraints.
6. Check Constraint.

- **Domain constraints** : specify that within each tuple, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$.
- The domain integrity constraint specifies that each attribute must have values derived from a valid range.

Concept of Key

- Key is an attribute or group of attributes, which is used to identify a row in a relation.
- Key can be broadly classified into :
 - (1) Superkey
 - (2) Candidate key
 - (3) Primary key



- **Key Constraints** : By definition of relation, all elements of a set of tuples are distinct;
 - This means that no two tuples can have the same combination of values for *all their* attributes.
- Usually, there are other **subsets of attributes of a relation schema R** *with* the property that no two tuples in any relation state $r(R)$ *should have the same* combination of values for these attributes.
- If we denote one such subset of attributes by SK; then for any two *distinct tuples $t1$ and $t2$ in $r(R)$* , we have the constraint that:

$$t1[SK] \neq t2[SK]$$

- **Super key** of R: A set of attributes SK of R specifies a uniqueness constraint that no two tuples *in any valid relation instance* $r(R)$ will have the same value for SK.
- A superkey is a subset of attributes of an entity-set that uniquely identifies the entities.
- Superkeys represent a constraint that prevents two entities from ever having the same value for those attributes.

Eg: Customer (Cname, Cid, CSSN, Address, DoB)

- A certain set of columns that ensures the uniqueness of each customer.
 - Cname+CSSN+DoB
 - Cid+Cname+CSSN

Eg: Consider the STUDENT relation.

- The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn.
- Any set of attributes that includes Ssn:
- Eg: {Ssn, Name, Age}—is a superkey.
- However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.
- In general, any superkey formed from a single attribute is also a key.
- A key with multiple attributes must require *all* its attributes together to have the uniqueness property.

- **Candidate Key** : is a minimal superkey.
 - that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint
- A candidate key for a relation schema is a minimal set of attributes whose values uniquely identify tuples in the corresponding relation.
- Essentially, a Superkey is a Candidate key with extra columns in it.

- The **primary key** is simply a candidate key chosen to be the main key.
- If a particular attribute is the primary key, then:
 1. that attribute value cannot be NULL.
 2. Also it has to be distinct.
- Non-primary candidate keys are also known as **alternative keys**.

Eg: Employee relation with attributes : Eid, Ename, Eage, Eexperience and salaryrelation:

- The Superkeys can be Eid, Ename, Eage, Eexperience etc.
- Candidate keys can be Eid, Ename, Eage
- Primary key is Eid.

Implementation of constraints in Oracle

- You can define constraints syntactically in two ways:
 1. **Inline** specification
 2. **Out-of-line** specification.
- 1. As part of the definition of an individual column or attribute.
- 2. As part of the table definition.
- NOT NULL constraints must be declared inline.
- All other constraints can be declared either inline or out of line.

● The six types of integrity constraints are:

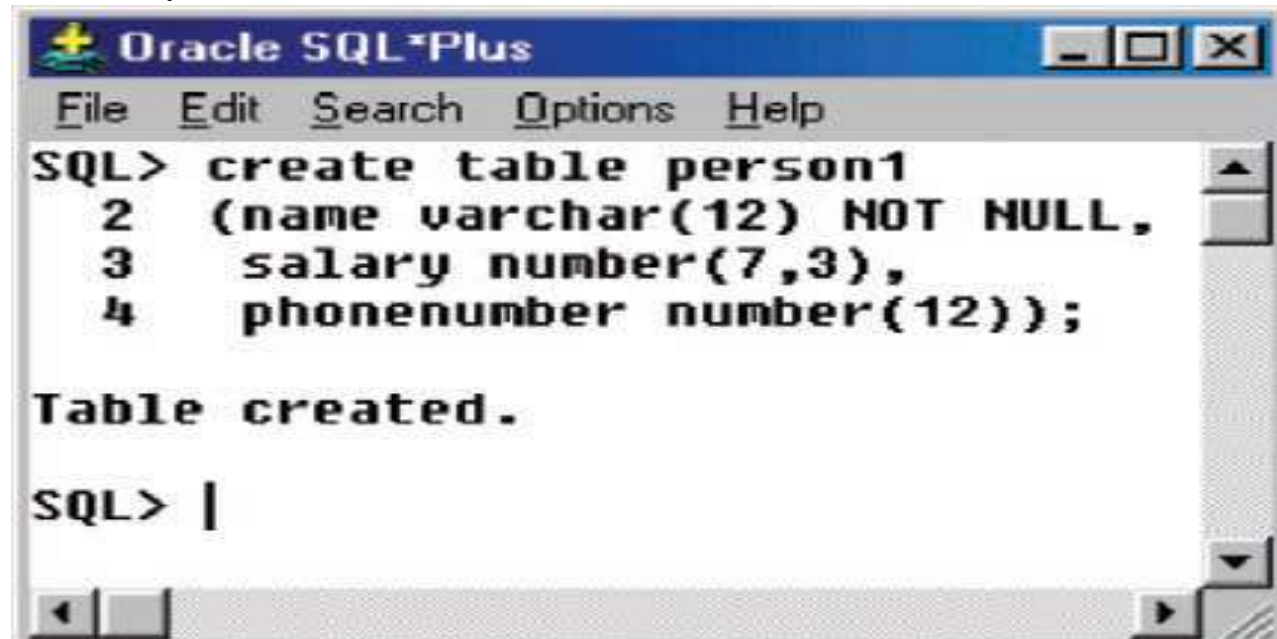
- NOT NULL constraint
- unique constraint
- primary key constraint
- foreign key constraint
- check constraint

- The syntax of NOT NULL constraint :

CREATE TABLE table name

(column name1, data-type of the column1, **NOT NULL**
column name2, data-type of the column2,
column nameN, data-type of the columnN);

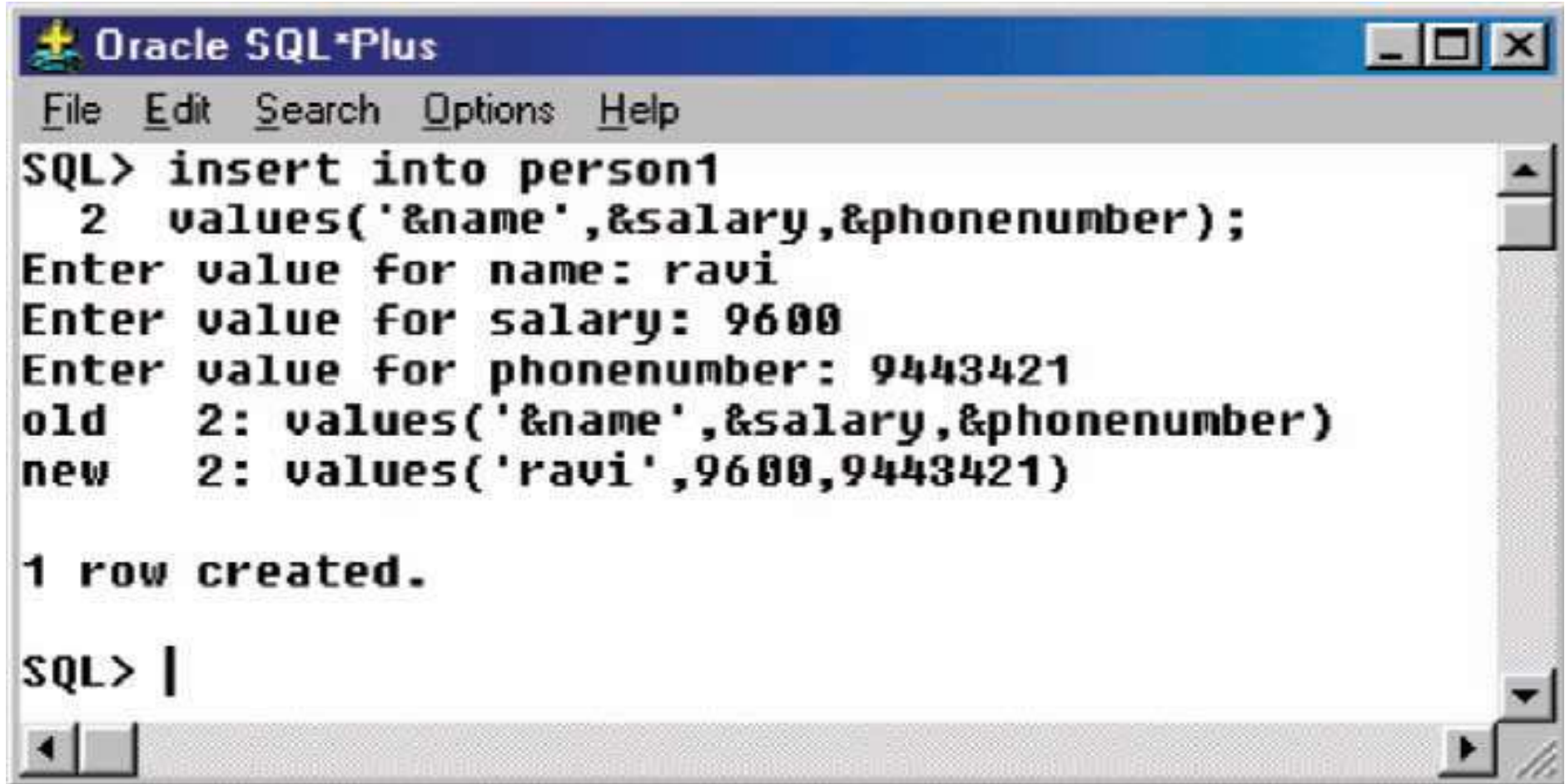
- The above syntax indicates that column1 is declared as NOT NULL.



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> create table person1
2  (name varchar(12) NOT NULL,
3   salary number(7,3),
4   phonenumber number(12));

Table created.

SQL> |
```

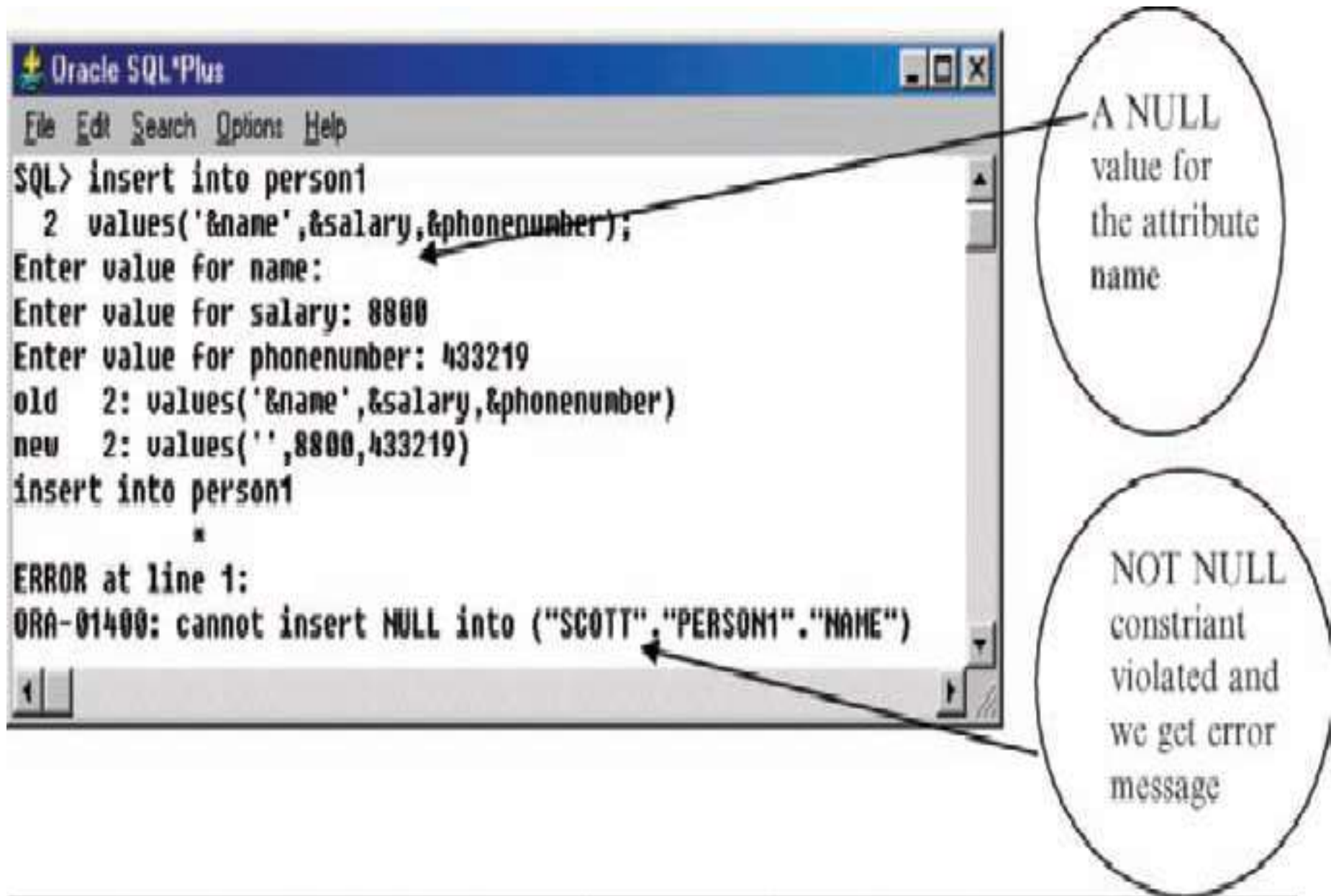
The image shows a screenshot of the Oracle SQL*Plus application window. The title bar at the top reads "Oracle SQL*Plus" and includes standard window control buttons (minimize, maximize, close). Below the title bar is a menu bar with the following options: File, Edit, Search, Options, and Help. The main text area contains the following text:

```
SQL> insert into person1
      2  values('&name',&salary,&phonenum);
Enter value for name: ravi
Enter value for salary: 9600
Enter value for phonenum: 9443421
old      2: values('&name',&salary,&phonenum)
new      2: values('ravi',9600,9443421)

1 row created.

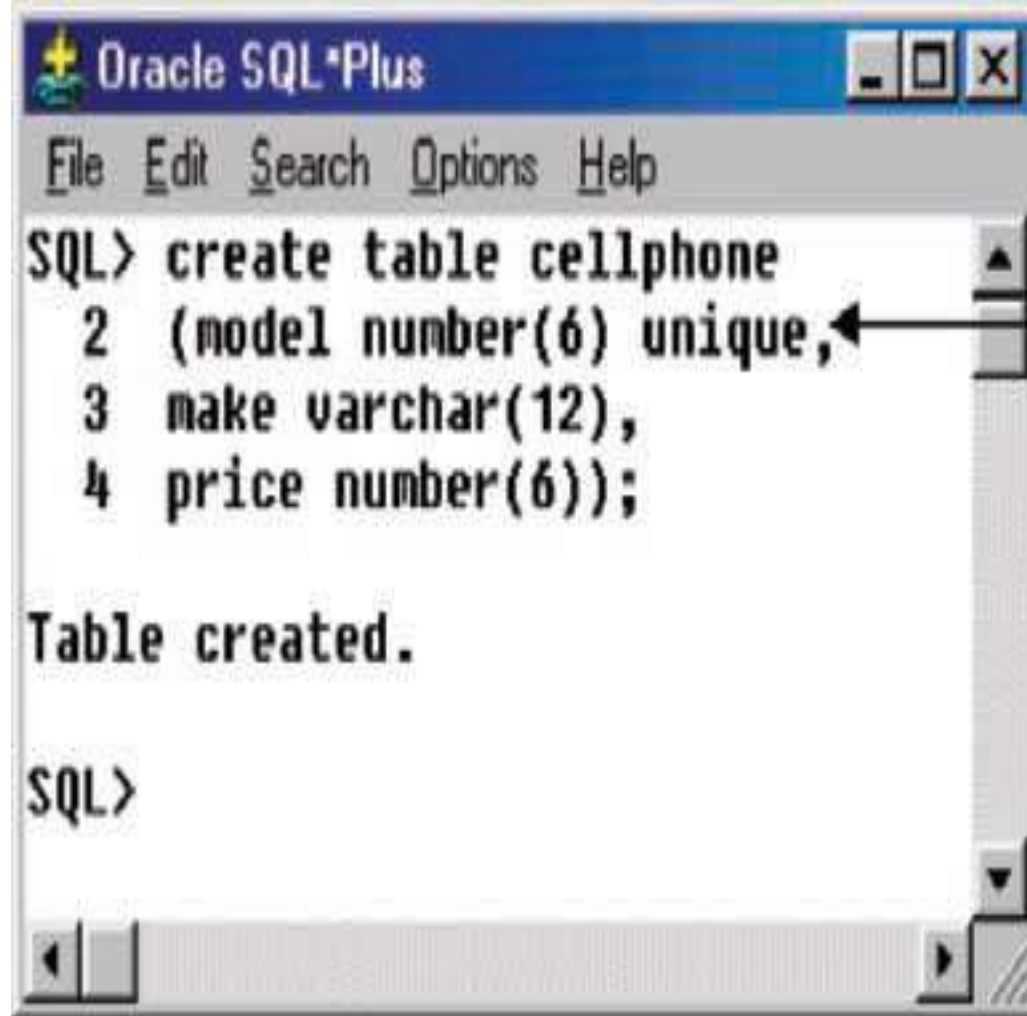
SQL> |
```

At the bottom of the window, there are navigation buttons: a left arrow, a right arrow, and a double right arrow.



- A **unique constraint** prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.
- A **composite unique key** designates a combination of columns as the unique key.
 - When you define a unique constraint inline, you need only the UNIQUE keyword.
 - When you define a unique constraint out of line, you must also specify one or more columns. You must define a composite unique key out of line.

- To satisfy a unique constraint, no two rows in the table can have the same value for the unique key.
 - However, the unique key made up of a single column can contain nulls.
 - To satisfy a composite unique key, no two rows in the table or view can have the same combination of values in the key columns.
- Any row that contains nulls in all key columns automatically satisfies the constraint.
- However, two rows that contain nulls for one or more key columns and the same combination of values for the other key columns violate the constraint.



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> create table cellphone
2  (model number(6) unique,
3  make varchar(12),
4  price number(6));

Table created.

SQL>
```

Unique constraint
is imposed on the
attribute model

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> select *
  2  from cellphone;

  MODEL MAKE          PRICE
-----
  1100 nokia          4000
  3300 nokia          3500
  6610 nokia          9000

SQL> |
```

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> insert into cellphone
  2  values(&model,'&make',&price);
Enter value for model: 1100
Enter value for make: nokia
Enter value for price: 3000
old   2: values(&model,'&make',&price)
new   2: values(1100,'nokia',3000)
insert into cellphone
      *
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C00820) violated

SQL> |
```


Difference Between NOT NULL and UNIQUE Constraint

- The unique constraint accepts NULL values
- NOT NULL constraint will not accept NULL values.
- Note NOT NULL constraint accepts duplicate values, whereas UNIQUE constraint will not accept duplicate values.

- A **primary key** constraint designates a column as the primary key of a table or view.
- A **composite primary key** designates a combination of columns as the primary key.
- When you define a primary key constraint inline, you need only the PRIMARY KEY keyword.
- When you define a primary key constraint out of line, you must also specify one or more columns. You must define a composite primary key out of line.

- A primary key constraint combines a NOT NULL and UNIQUE constraint in one declaration.
- Therefore, to satisfy a primary key constraint:
 - No primary key value can appear in more than one row in the table.
 - No column that is part of the primary key can contain a null.

- **Syntax to define a Primary key at column level:**

- column name datatype [CONSTRAINT constraint_name]
PRIMARY KEY

- **Syntax to define a Primary key at table level:**

- [CONSTRAINT constraint_name] PRIMARY KEY
(column_name1,column_name2,..)

Eg: CREATE TABLE locations_demo

```
( location_id NUMBER(4) CONSTRAINT loc_id_pk PRIMARY  
KEY ,  
street_address VARCHAR2(40) ,  
postal_code VARCHAR2(12) ,  
city VARCHAR2(30) ,          state_province VARCHAR2(25) ,  
country_id CHAR(2) ) ;
```

- The `loc_id_pk` constraint, specified inline, identifies the `location_id` column as the primary key of the `locations_demo` table.
- This constraint ensures that no two locations in the table have the same location number and that no location identifier is NULL.

● Out of line cnstraint specification.

```
Eg: CREATE TABLE locations_demo  
( location_id NUMBER(4) ,  
  street_address VARCHAR2(40) ,  
  postal_code VARCHAR2(12) ,  
  city VARCHAR2(30) ,  
  state_province VARCHAR2(25) ,  
  country_id CHAR(2) ,  
  CONSTRAINT loc_id_pk PRIMARY KEY (location_id));
```

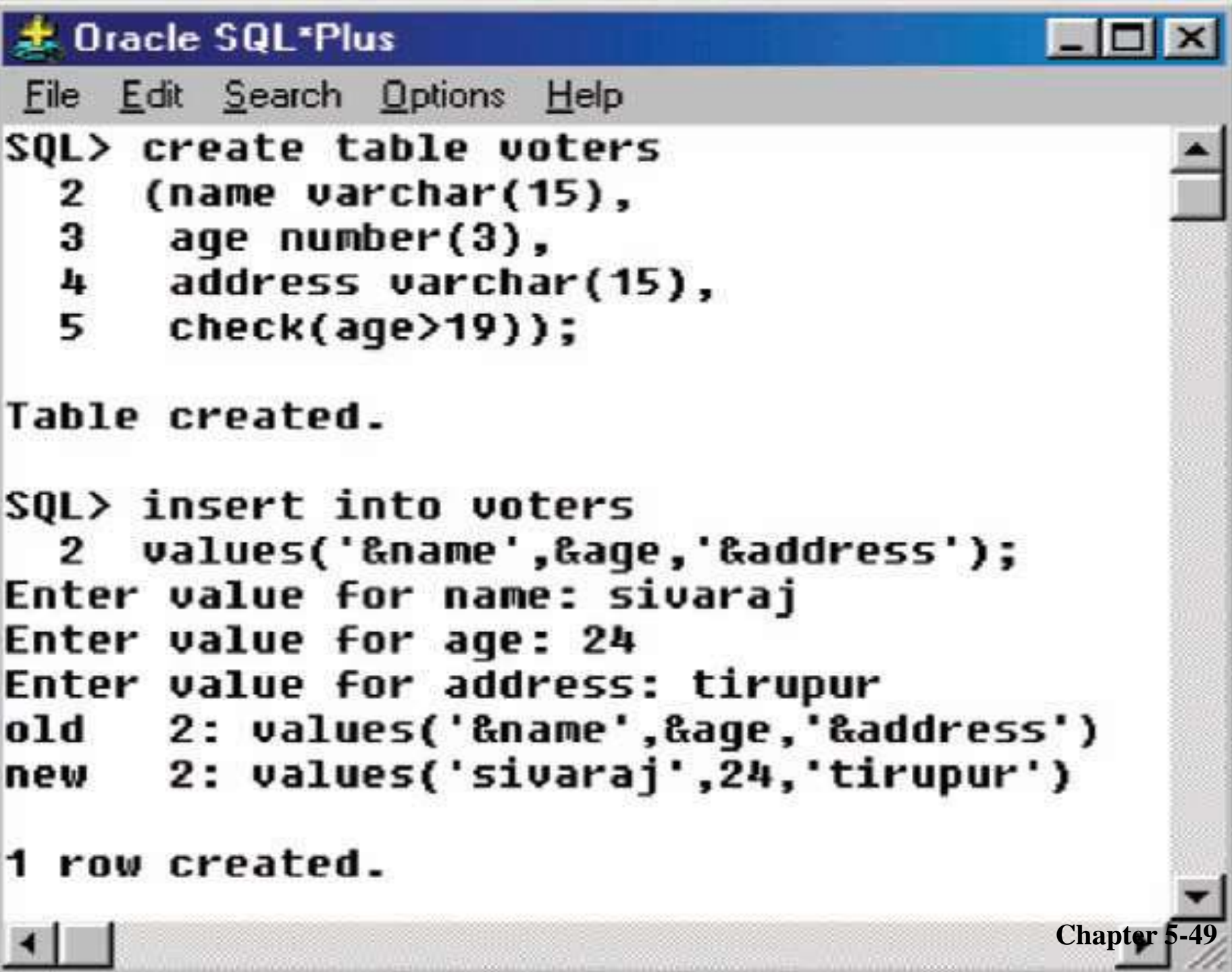
● Restrictions on Primary Key Constraints

- A table or view can have only one primary key.
- None of the columns in the primary key can be LOB, LONG, LONG RAW, BFILE, REF, TIMESTAMP WITH TIME ZONE, or user-defined type.
- However, the primary key can contain a column of TIMESTAMP WITH LOCAL TIME ZONE.
- The size of the primary key cannot exceed approximately one database block.
- A composite primary key cannot have more than 32 columns.

- You cannot designate the same column or combination of columns as both a primary key and a unique key.
- You cannot specify a primary key when creating a subview in an inheritance hierarchy.
 - The primary key can be specified only for the top-level (root) view.

CHECK Constraint

- A **check constraint** lets you specify a condition that each row in the table must satisfy.
- To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).
- When Oracle evaluates a check constraint condition for a particular row,
 - any column names in the condition refer to the column values in that row.
- Attribute value check is checked only when the value of the attribute is inserted or updated.



The image shows a screenshot of the Oracle SQL*Plus command window. The title bar at the top reads "Oracle SQL*Plus" and includes standard window control buttons (minimize, maximize, close). The menu bar below the title bar contains "File", "Edit", "Search", "Options", and "Help". The main text area displays the following SQL commands and their output:

```
SQL> create table voters
      2  (name varchar(15),
      3    age number(3),
      4    address varchar(15),
      5    check(age>19));

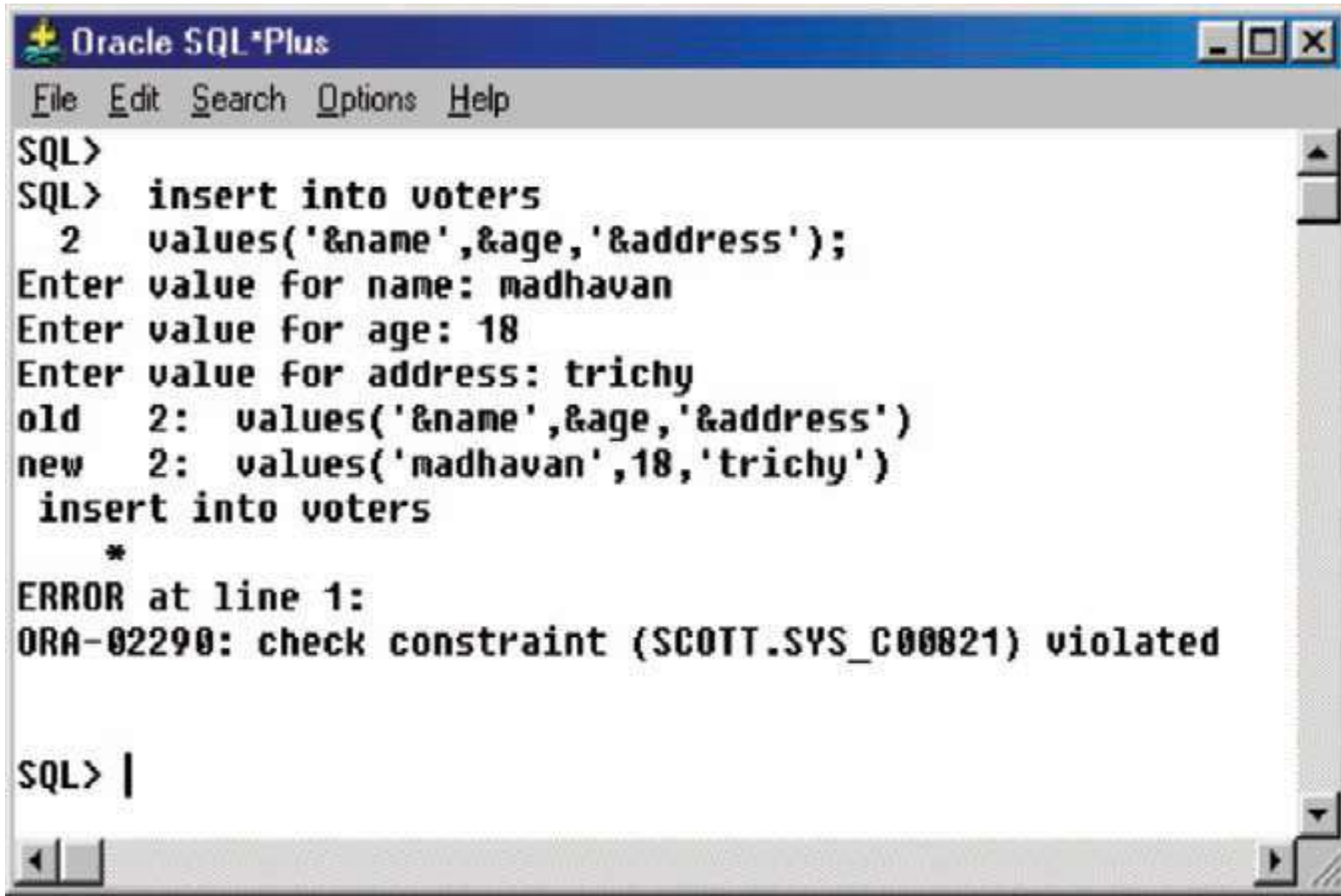
Table created.

SQL> insert into voters
      2  values('&name',&age,'&address');
Enter value for name: sivaraj
Enter value for age: 24
Enter value for address: tirupur
old   2: values('&name',&age,'&address')
new   2: values('sivaraj',24,'tirupur')

1 row created.
```

At the bottom right of the window, the text "Chapter 5-49" is visible.

● Check constraint violation

A screenshot of the Oracle SQL*Plus command-line interface. The window has a title bar with the Oracle logo and the text "Oracle SQL*Plus". Below the title bar is a menu bar with "File", "Edit", "Search", "Options", and "Help". The main text area shows the following sequence of commands and responses:
SQL>
SQL> insert into voters
2 values('&name',&age,'&address');
Enter value for name: madhavan
Enter value for age: 18
Enter value for address: trichy
old 2: values('&name',&age,'&address')
new 2: values('madhavan',18,'trichy')
insert into voters
*
ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C00821) violated

SQL> |
The window includes standard scroll bars on the right and bottom edges.

- A **foreign key** is a key used to link two tables together. This is sometimes called a **referencing key**.
- Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

- The following SQL creates a FOREIGN KEY :
 - CREATE TABLE Orders (O_Id int NOT NULL PRIMARY KEY, OrderNo int NOT NULL, P_Id int REFERENCES Persons(P_Id));
- If the table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by [altering a table](#).
 - ALTER TABLE ORDERS ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
- To drop a FOREIGN KEY constraint:
 - ALTER TABLE ORDERS DROP FOREIGN KEY;

- **Constraints on NULL:** constraint on attributes specifies whether NULL values are permitted or not.

Eg: If every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

- **Integrity constraints** provide a way to ensure that changes made to the database by authorized users do not result in a loss of data consistency.
- A **relational database** usually contains many relations, with tuples in relations that are related in various ways.
- A **relational database schema** S is a set of relation schemas $S = \{R1, R2, ..., Rm\}$ and a set of integrity constraints IC.

- A **relational database state** DB of S is a set of relation states $DB = \{r_1, r_2, \dots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC.
- Relational database schema for COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.
- The underlined attributes represent primary keys.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

Schema diagram for
COMPANY relational
database schema.

- A database state that does not obey all the integrity constraints is called an **invalid state**, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **valid state**.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son

- The **entity integrity constraint** states that no primary key value can be NULL.

$t[\text{PK}] \neq \text{null}$ for any tuple t in $r(R)$

Reason: primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples.

Eg: if two or more tuples had NULL for their primary keys, distinguishing them while referencing from other relations will be difficult.

Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

- Key constraints and entity integrity constraints are specified on individual relations.

Referential Integrity

- In the relational data model, associations between tables are defined through the use of **foreign keys**.
- The referential integrity rule states that a database must not contain any unmatched foreign key values.
- The referential integrity rule does not imply a foreign key cannot be null.
- There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null.
- A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

- The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2 .
- A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following rules:
 1. The attributes in FK have the same domain(s) as the PK of R_2 ; the attributes FK are said to **reference** or **refer to** the relation R_2 .
 2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL.
- In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 **references** or **refers** to the tuple t_2 .

- In this definition, R1 is called the referencing relation and R2 is the referenced relation.
- If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.

Other Types of Constraints

- Another type of constraint is the **functional dependency constraint**, which establishes a functional relationship among two sets of attributes X and Y.
- This constraint specifies that the value of X determines a unique value of Y in all states of a relation; it is denoted as a functional dependency $X \rightarrow Y$.

Semantic Integrity Constraints:

- Such constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose **constraint specification language**.

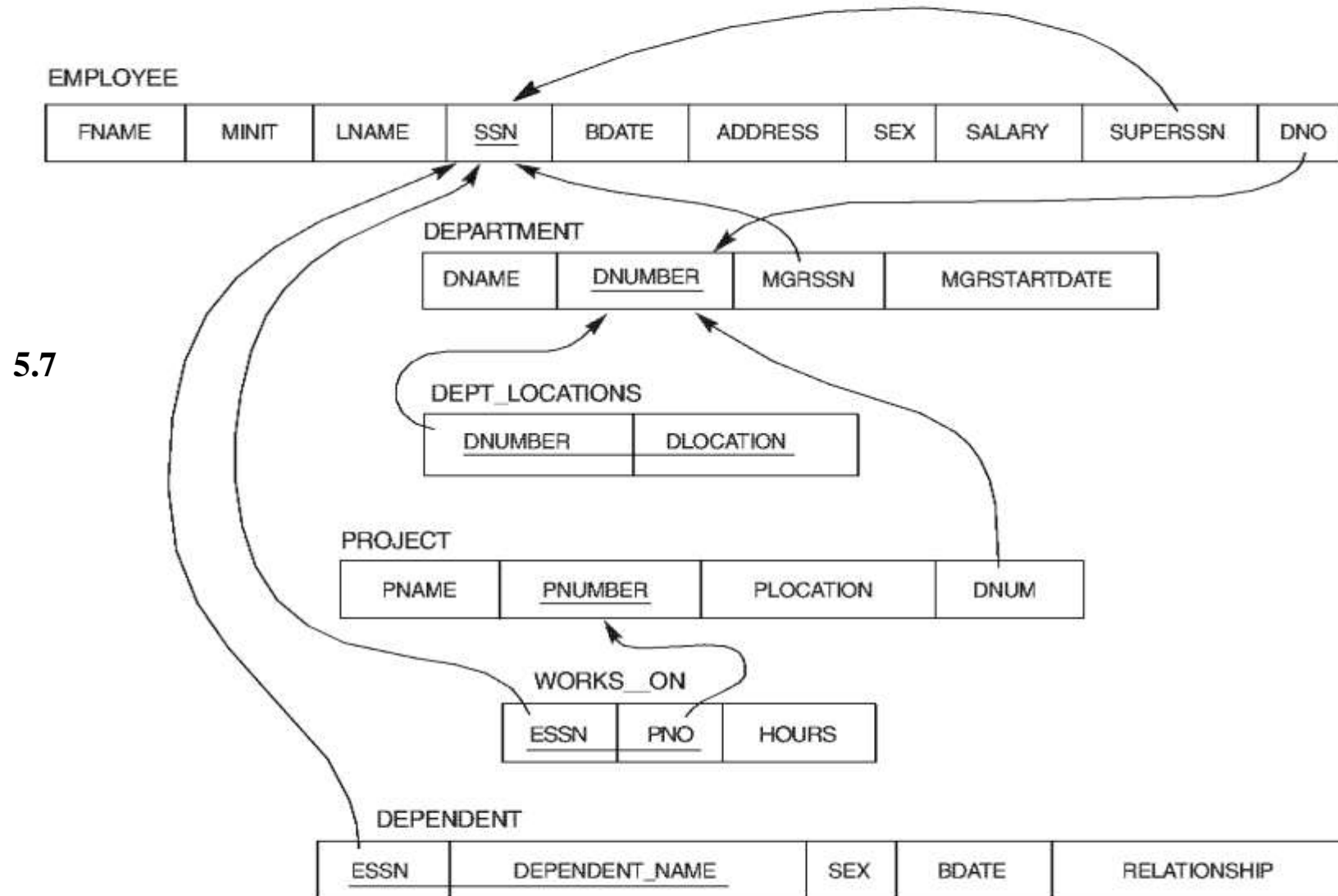
E.g. “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”

- SQL-99 allows triggers and assertions for this.
- Another type of constraint, called **transition constraints**, can be defined to deal with state changes in the database.

Eg: “the salary of an employee can only increase.”

- Such constraints are typically enforced by the application programs or specified using active rules and triggers.

Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.



Update Operations on Relations

1. INSERT a tuple.
 2. DELETE a tuple.
 3. MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
 - Several update operations may have to be grouped together.
 - Updates may *propagate* to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
 - Cancel the operation that causes the violation (REJECT option)
 - Perform the operation but inform the user of the violation
 - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
 - Execute a user-specified error-correction routine