chapter 7:

INTRODUCTION TO SQL: Data Definition and Data Manipulation in SQL

Dr. Linda Mahmoudi

SQL Overview

- **Originally "Structured Query Language", today a proper name**
- A language with several functionalities
  - comprises both DDL and DML
- **The standard for relational database management systems (RDBMS)**
- **RDBMS:** A database management system that manages data as a collection of tables in which all relationships are represented by common values in related tables

SQL Environment

- Catalog
  - A set of schemas that constitute the description of a database
- Schema
  - The structure that contains descriptions of objects created by a user (base tables, views, constraints)
- Data Definition Language (DDL)
  - Commands that define a database, including creating, altering, and dropping tables and establishing constraints
- Data Manipulation Language (DML)
  - Commands that maintain and query a database
- Data Control Language (DCL)
  - Commands that control a database, including administering privileges and committing data

SQL environment

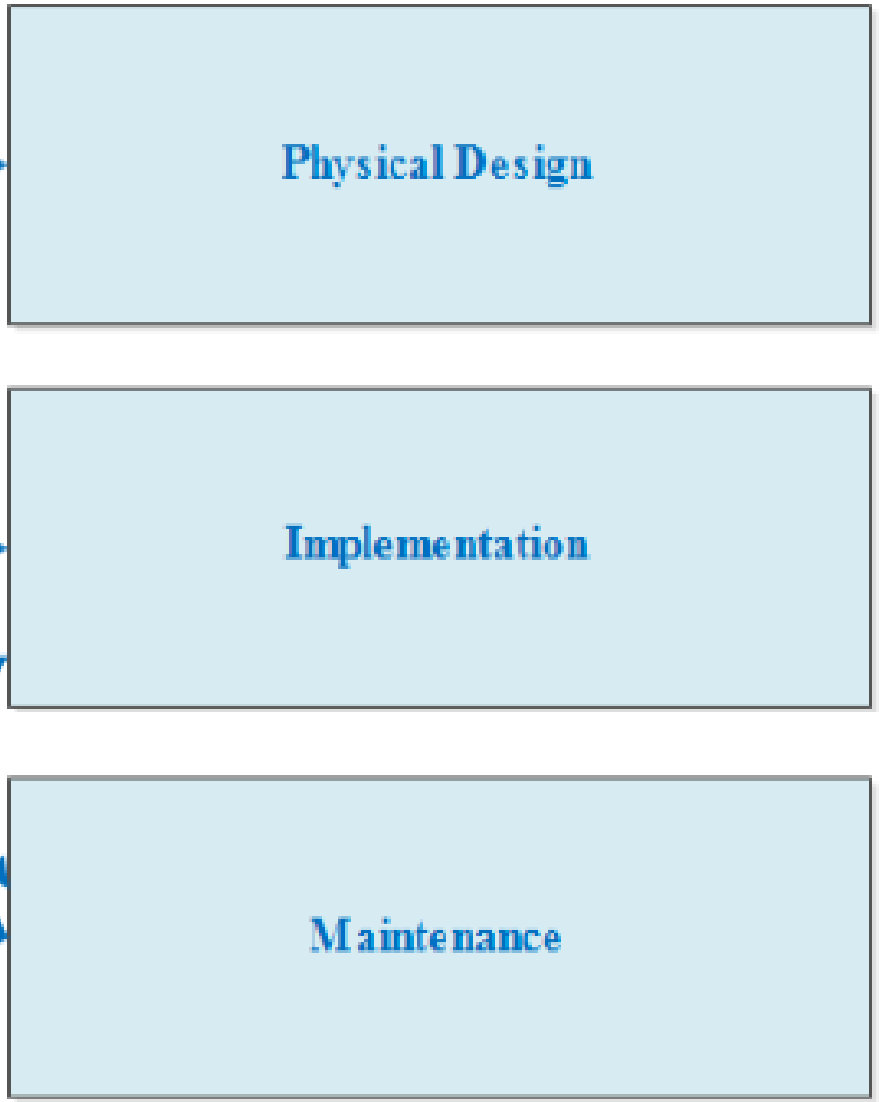
Types of Commands

database development process.

Data definition language (DDL)
Commands used to define a database, including those for creating, altering, and dropping tables and establishing constraints. Commands: Create, Alter, Drop, Trancate, Rename.

Data manipulation language (DML)
Commands used to maintain and query a database, including those for updating, inserting, modifying, and querying data. Commands: commands: Insert, Update, Delete,Select

Data control language (DCL)
Commands used to control a database, including those for administering privileges and committing (saving) data. Commands: GRANT, REVOKE, ADD



Data Definition Language

Three Commands:

- CREATE
- ALTER
- DROP

SQL Data Types

Data type	Description
Char	Stores string values containing any characters in a character set. CHAR is defined to be a fixed length.
Varchar2(n) Varchar(n)	Stores string values containing any characters in a character set but of definable variable length.
BLOB (Binary Large Object)	Stores binary string values in hexadecimal format. BLOB is defined to be a variable length. (Oracle also has CLOB and NCLOB, as well as BFILE for storing unstructured data outside the database.)
Number[(n,m)]	Stores exact numbers with a defined precision and scale.
Integer[(n)]	Stores exact numbers with a predefined precision and scale of zero
Date	Stores information of date/hour data type
Boolean	Stores truth values: TRUE, FALSE, or UNKNOWN.

SQL Constraints

Constraint	Description
Not Null	prohibits a database value from being null. Is defined at the column-constraint level. Example: Emp_ Name Varchar2(25) NOT NULL
Unique	prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null. Is defined at either the table or column-constraint level
Check	requires a value in the database to comply with a specified condition. Is defined at either the table or column-constraint level Syntax: Column level: Attribute_Name Data Type CONSTRAINT constraint_name check(condition) Table level: …., CONSTRAINT constraint_name check(condition)

SQL Constraints(cont.)

Constraint	Description
primary key	A **primary key constraint** combines a NOT NULL constraint and a unique constraint in a single declaration. That is, it prohibits multiple rows from having the same value in the same column or combination of columns and prohibits values from being null. Is defined at either the table or column-constraint level
foreign key	A foreign key is a way to enforce referential integrity within your Oracle database. A **foreign key constraint** requires values in one table to match values in another table. Is defined at either the table or column-constraint level Syntax: Attribute_Name Data type [CONSTRAINT constraint_name] foreign key REFERENCES table_Name(refattribute_Name) [ON DELETE CASCADE| ON DELETE SET NULL|SET DEFAULT ON UPDATE CASCADE]

SQL Constraints(cont.)

Constraint	Description
foreign key	A foreign key is a way to enforce referential integrity within your Oracle database. A **foreign key constraint** requires values in one table to match values in another table. Is defined at either the table or column-constraint level ❑ Restricting: ▫ Deletes of primary records ▫ Updates of primary records ▫ Inserts of dependent records Syntax: Attribute_Name Data type [CONSTRAINT constraint_name] foreign key REFERENCES table_Name(refattribute_Name) [ON DELETE CASCADE| ON DELETE SET NULL|SET DEFAULT ON UPDATE CASCADE]

SQL Constraints(cont.)

- Column-constraint level

```
column [CONSTRAINT constraint_name]
constraint_type,
```

- Table-constraint level

```
column,...
   [CONSTRAINT constraint_name] constraint_type
   (column, ...),
```

Steps in Table Creation

1. Identify data types for attributes
2. Identify columns that can and cannot be null
3. Identify columns that must be unique (candidate keys)
4. Identify primary key–foreign key mates
5. Determine default values
6. Identify constraints on columns (domain specifications)
7. Create the table and associated indexes

Create Tables

the most important command of the DDL in SQL is

create table

– Defines a relation schema (with attributes and integrity constraints)

– Creates an empty instance of the schema

Note: Each SQL command ends with a semicolon (;).

Syntax:

create table *TableName* **(**
Attribute1_Name Data type [*Constraint*],
.....
AttributeN_Name Data type [*Constraint*],
[*OtherConstraints*]**);**

Example:

```
CREATE TABLE suppliers (
        supplier_id number(10) NOT NULL Primary key,
        supplier_namevarchar2(50) NOT NULL,
        city varchar2(50),
        state varchar2(25),
        zip_codevarchar2(10) );
```

```
CREATE TABLE suppliers (
        supplier_id number(10) NOT NULL,
        supplier_namevarchar2(50) NOT NULL,
        ....
        zip_codevarchar2(10),
        Primary key (supplier_id));
```

Example:

```
CREATE TABLE customers(
                customer_id number(10) NOT NULL,
                customer_name varchar2(50) NOT NULL,
                city varchar2(50),
                Dept character(15),
                Salary numeric(9) default 0,
                City character(15),
        CONSTRAINT customers_pk PRIMARY KEY (customer_id)
        );
```

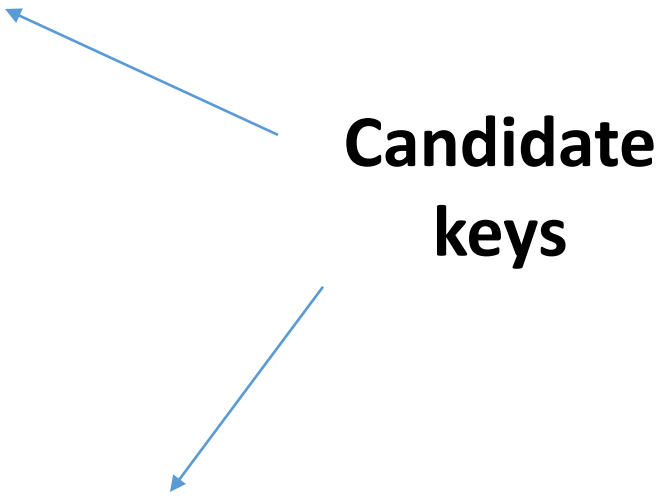
Example:

```
CREATE TABLE employees (
                employee_number number(10) NOT NULL,
                employee_name varchar2(50) NOT NULL,
                department_id number(10),
                salary number(6) check(salary between ,
        CONSTRAINT employees_pk PRIMARY KEY (employee_number),
        CONSTRAINT fk_departments FOREIGN KEY (department_id)
        REFERENCES departments (department_id) );
```

```
create table Employee ( ...
   FirstName  character(20) not null,
   LastName   character(20) not null,
   unique (LastName,FirstName)
);
```

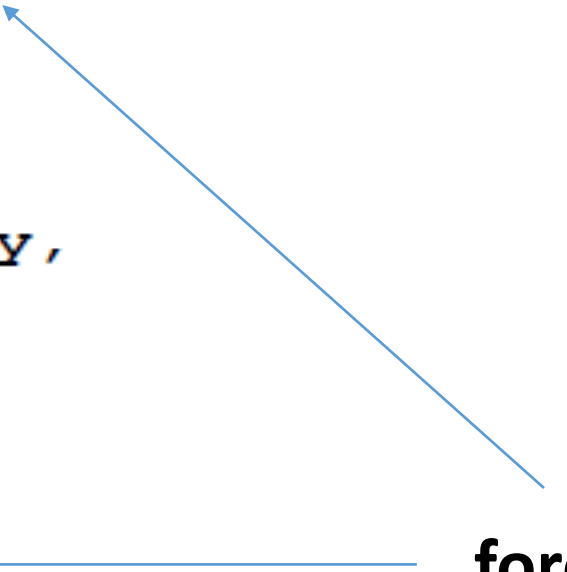
is different from:

```
create table Employee ( ...
   FirstName  character(20) not null unique,
   LastName   character(20) not null unique
);
```



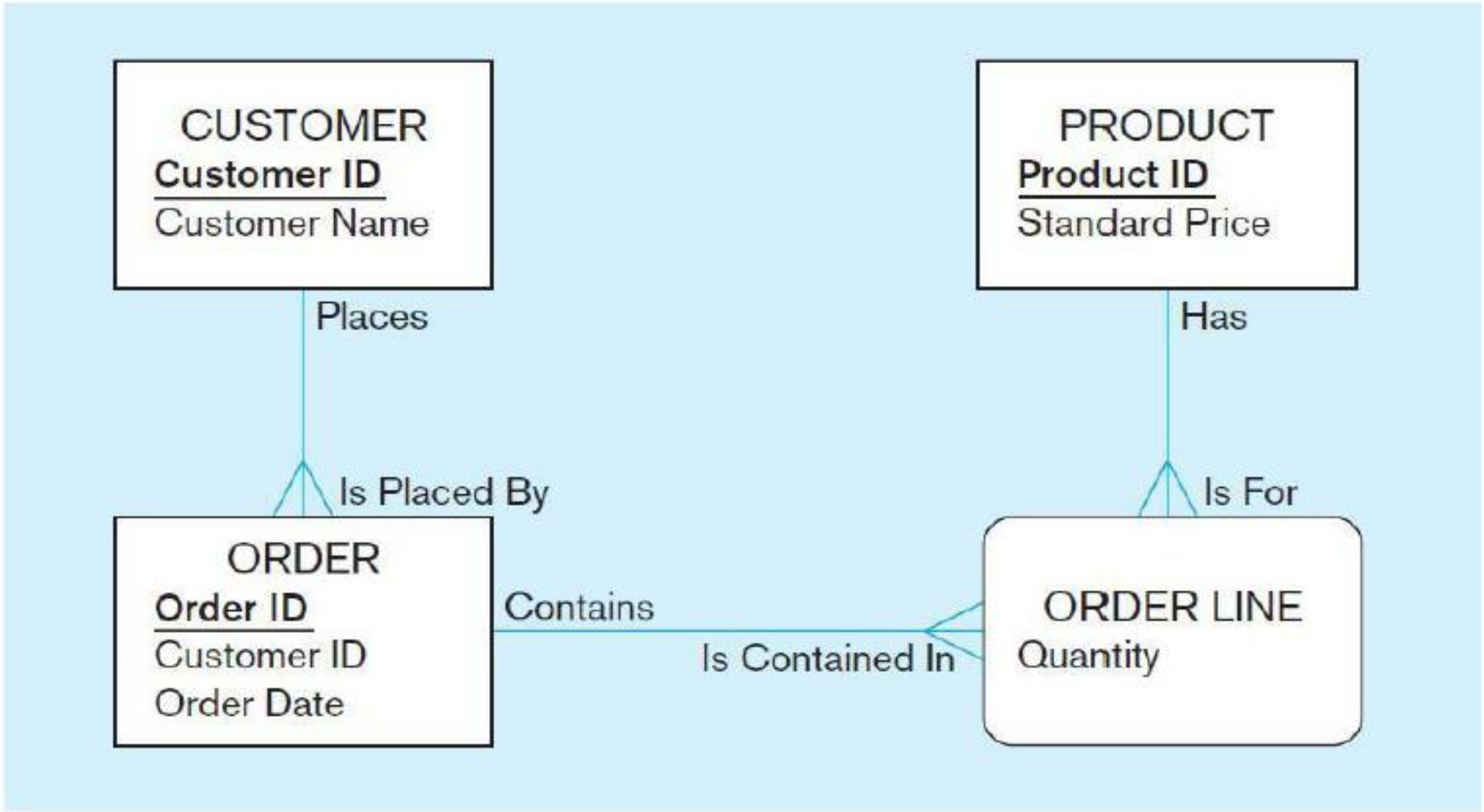
```
create table Student(
 StudNo   character(10) primary key,
 Name     character(20),
 Hons     character(3),
 Tutor    character(20) references Staff(Lecturer),
 Year     smallint);


create table Staff(
 Lecturer  character(20) primary key,
 RoomNo    character(4),
 Appraiser character(20),
 foreign key (Appraiser) references
              Staff(Lecturer)
              on delete set null
              on update cascade);
```



foreign keys

Create Table- Example



SQL database definition commands

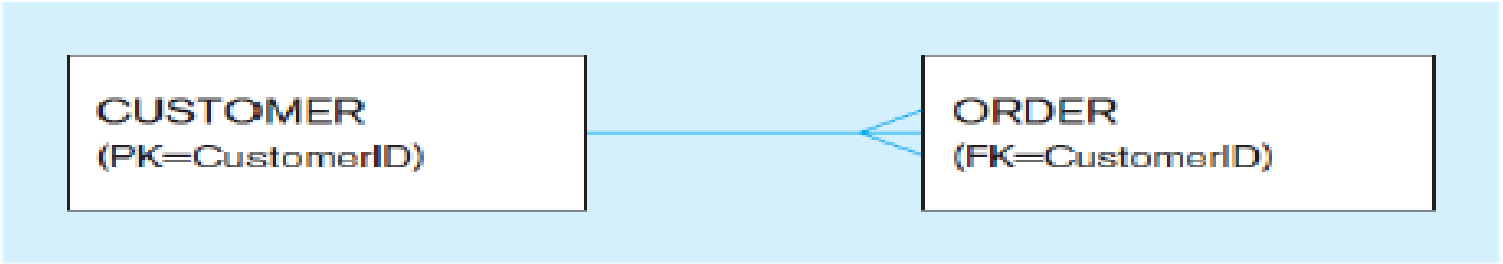
```
CREATE TABLE Customer_T
                        (CustomerID                     NUMBER(11,0)        NOT NULL,
                         CustomerName                   VARCHAR2(25)        NOT NULL,
                         CustomerAddress                VARCHAR2(30),
                         CustomerCity                   VARCHAR2(20),
                         CustomerState                  CHAR(2),
                         CustomerPostalCode             VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));

CREATE TABLE Order_T
                       (OrderID                         NUMBER(11,0)        NOT NULL,
                        OrderDate                       DATE DEFAULT SYSDATE,
                        CustomerID                      NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
                       (ProductID                       NUMBER(11,0)        NOT NULL,
                        ProductDescription              VARCHAR2(50),
                        ProductFinish                   VARCHAR2(20)
                                    CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                                             'Red Oak', 'Natural Oak', 'Walnut')),
                        ProductStandardPrice            DECIMAL(6,2),
                        ProductLineID                   INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));

CREATE TABLE OrderLine_T
                       (OrderID                         NUMBER(11,0)        NOT NULL,
                        ProductID                       INTEGER             NOT NULL,
                        OrderedQuantity                 NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

Ensuring data integrity through updates



Restricted Update: A customer ID can only be deleted if it is not found in ORDER table.

```
CREATE TABLE CustomerT
                (CustomerID          INTEGER DEFAULT '999'     NOT NULL,
                 CustomerName        VARCHAR(40)               NOT NULL,
                 . . .
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID),
ON UPDATE RESTRICT);
```

Cascaded Update: Changing a customer ID in the CUSTOMER table will result in that value changing in the ORDER table to match.

```
. . . ON UPDATE CASCADE);
```

Set Null Update: When a customer ID is changed, any customer ID in the ORDER table that matches the old customer ID is set to NULL.

```
. . . ON UPDATE SET NULL);
```

Set Default Update: When a customer ID is changed, any customer ID in the ORDER tables that matches the old customer ID is set to a predefined default value.

```
. . . ON UPDATE SET DEFAULT);
```

Relational integrity is enforced via the primary-key to foreign-key match

Policies

- Determine the effect of `delete` and `update` statements

- Syntax

  `on delete` *Action*

  where *Action* can be

`cascade`	(propagate the deletion)
`restrict`	(do nothing if the row is referenced)
`no action`	(as `restrict`, but return an error)
`set default`	
`set null`	

- The same actions exist for updates (`on update`)

CREATE TABLE AS Statement

- CREATE TABLE AS statement is used to create a table from an existing table by copying the existing table's columns.
- Syntax:

```
SQL> CREATE TABLE new_table
AS SELECT Statement;
```

ALTER TABLE Statement

- **ALTER TABLE statement** is used to:
  - Add a new column|COnstraint
  - Modify an existing column
  - Define a default value for the new column
  - Drop a column|Constraint
- Syntax:

```
ALTER TABLE table_name alter_table_action;
```

  - Some of the alter_table_actions available are:

```
ADD [COLUMN] column_definition
ALTER [COLUMN] column_name SET DEFAULT default-value
ALTER [COLUMN] column_name DROP DEFAULT
DROP [COLUMN] column_name [RESTRICT] [CASCADE]
ADD table_constraint
```

ALTER TABLE Statement(cont.)

- **Add column in table**
  - Syntax:

```
ALTER TABLE table_name
ADD column_name column_definition;
```

- **Add multiple columns in table**
  - **Syntax:**

```
ALTER TABLE table_name
ADD (column_1 column_definition,
column_2 column_definition,
...
column_n column_definition);
```

ALTER TABLE Statement(cont.)

- **Modify column in table**
  - Syntax:

```
ALTER TABLE table_name
MODIFY column_name column_type;
```

- **Modify Multiple columns in table**
  - **Syntax:**

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

ALTER TABLE Statement(cont.)

- **Drop column in table**
  - Syntax:

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

- **Rename column in table (NEW in Oracle 9i Release 2)**
  - **Syntax:**

```
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

ALTER TABLE Statement(cont.)

- **Rename table**
  - Syntax:

```
ALTER TABLE table_name RENAME TO
new_table_name;
```

- **Add Table constraint**
  - **Syntax:**

```
ALTER TABLE table_name
Add CONSTRAINT constraint_name
Constraint_Type(Attribute(s);
```

Example: ALTER TABLE

Alter table vendor
Modify Phone Number Varchar2(40);

- To delete the constraint not null from attribute phone number

Alter table Vendor
Drop Constraint Not Null(phone number);

- To drop column Phone number from vendor table

Alter table Vendor
Drop Column Phone number;

Alter table Part
Add(Weight number Not null,
Category varchar2(25) not null);

Alter table Vendor
Add PhoneNumber Varchar2(15) not Null;

DROP TABLE Statement

- **DROP TABLE** statement allows you to remove tables from your schema:
- Syntax:

```
DROP TABLE Table_Name
[ CASCADE CONSTRAINTS ];
```

- **CASCADE CONSTRAINTS** optional. If specified, all referential integrity constraints will be dropped as well.
- Example:
  - DROP TABLE CUSTOMER_T

Example: DROP TABLE

- To delete a table:

Drop Table vendor cascade constraints;

-- Remove the table vendor and the constraint referential

integrity with the table part_supplier

Data Manipulation Language

Three Commands:

- Insert
- Delete
- Update

Insert Statement

insert into *Table* [**(** *Attributes* **)**]

values(*Values* **)** *(values are stated explicitly)*

or

insert into *Table* [**(** *Attributes*)]

select … *(values are produced by a query)*

Insert Statement

- Adds one or more rows to a table
  - Syntax:

```
INSERT INTO table [(column1, column2, ... column_n ) ]
VALUES (expression1, expression2, ... expression_n );
```

- Inserting from another table
  - Syntax:

```
INSERT INTO table (column1, column2, ... column_n )
SELECT expression1, expression2, ... expression_n
FROM source_table [WHERE conditions];
```

Example: Insert Statement

MotherChild

mother	child
Lisa	Mary
Lisa	Greg
Anne	Kim
Anne	Phil
Mary	Andy
Mary	Rob

FatherChild

father	child
Steve	Frank
Greg	Kim
Greg	Phil
Frank	Andy
Frank	Rob

Person

name	age	income
Andy	27	21
Rob	25	15
Mary	55	42
Anne	50	35
Phil	26	30
Greg	50	40
Frank	60	20
Kim	30	41
Mike	85	35
Lisa	75	87

Example: Insert Statement

- insert into person values('Peter',25,52)
- insert into person(name, age, income) values('Paul',25,52)
- insert into person(name, income) values('Mary',55)
- insert into person (name)

  select father from fatherChild

  where father not in (select name from person)

Insertion: Comments

- The ordering of attributes in the attribute list (if present) and of the values in the value list is crucial
- The list of attributes and the list of values must have the same number of elements
- If the list of attributes is missing, the list of all attributes is taken, with the ordering taken from the table definition
- If the list of attributes does not contain all attributes of the relation, the default value or the value null (if possible) is inserted for the missing attributes

Delete Statement

- Removes rows from a table

Syntax:

delete from *Table* [**where** *Condition*]

Examples:

(conditions are similar to query conditions)

```
delete from person where age < 35

delete from fatherChild
where child not in (select name from person)
```

Elimination: Comments

- *All tuples* that satisfy the *condition* are eliminated
- May cause *eliminations in other relations* if the repair policy **`cascade`** has been specified for those relations
- Note: if the `where` part is *missing*, it is understood as `where true`

Update Statement

- Modifies data in existing rows

- **Syntax**:

```
update TableName
set  Attribute = < Expression | select ... |  null | default >
[ where Condition ]
```

- **Semantics**: all tuples of the table are modified that satisfy the **where** condition

- *Examples*:

```
update person set income = 45
where  name = 'Greg'

update person set income = income * 1.1
where  age < 30
```

DATA INTERROGATION LANGUAGE

SELECT Statement

The select Statement (Basic Version)

• **Query** statements in SQL start with the keyword

select

and return a result in table form

Select Attribute … Attribute

from Table … Table

[where Condition]

• The three parts are usually called

– target list

– from clause

– where clause

Example: Select Statement

- Name and income of persons that are less than 30:

Projection

$$\pi_{\text{name, income}}(\sigma_{\text{age<30}}(\text{Person}))$$

```
select name, income
from   person
where  age < 30
```

name	income
Andy	21
Rob	15
Phil	30

Naming Conventions

- To avoid ambiguities, every attribute name has two components

RelationName.AttributeName

- When there is no ambiguity, one can drop the initial component

RelationName .

```
select person.name, person.income
from   person
where  person.age < 30

can be written as:

select name, income
from   person
where  age < 30
```

select: Abbreviations

```
select name, income
from   person
where  age < 30
```

is an abbreviation for:

```
select person.name, person.income
from   person
where  person.age < 30
```

and also for:

```
select p.name as name,  p.income as income
from   person p
where  p.age < 30
```

Select Statement (cont.)

Clauses of the SELECT statement:

- **Select**: List the columns (and expressions) to be returned from the query
- **DISTINCT**: is used to remove duplicates from the result set
- **From**: Indicate the table(s) or view(s) from which data will be obtained
- **Where**: Indicate the conditions under which a row will be included in the result
- **Group By**: Indicate categorization of results
- **Having**: Indicate the conditions under which a category (group) will be included
- **Order BY**: Sorts the result according to specified criteria

SQL: Comparison Operators

- Comparison operators are used in the WHERE clause to determine which records to select. Here is a list of the comparison operators that you can use in SQL:

Comparison Operator	Description
=	Equal
<>	Not Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal