Suggested Books

1. Delphi/Kylix Database Development
   Eric Harmon
   Sams Publishing

2. Delphi in a Nutshell
   Ray Lischner
   O'Reilly Publishing

3. Delphi 6 Developer's Guide
   Xavier Pacheco et al
   Sams Publishing

What is SQL?

Introduction to SQL

Structured Query language-SQL
First studies at IBM Laboratories
First, it was developed in 1970 for IBM DB2
A Universal Language to be used in all major Database Management Systems

Introduction to SQL

12 Rules Defined by Dr.Codd for a Relational Database Management System

0. Databases should be administered by Relational Capabilities
1. Information Rule-All information in a RDBMS (including table and column names) should be stored as values of tables
2. Guaranteed Access-Each value in RDBMS should be guaranteed for access using primary key and/or column names
3. Systematic support for *null values*
4. Active, online Relational Catalog

"A Relational Model of Data for Large Shared Data Banks", 1970,Dr.E.F.Codd

12 Rules Defined by Dr.Codd for a Relational Database Management System

5. **A Data Language with a certain and clear syntax - This language should support data definition, manipulation, integrity rules, authorization and transaction processes**
6. **Rule of updating Views**
7. **Retrieving, inserting, updating and deleting data in set levels**
8. **Physical Data Independence- Applications should not be affected by the change of physical access and recording structures**
9. **Logical Data Independence- Applications should not be affected by the change in table structures**

"A Relational Model of Data for Large Shared Data Banks", 1970,Dr.E.F.Codd

Introduction to SQL

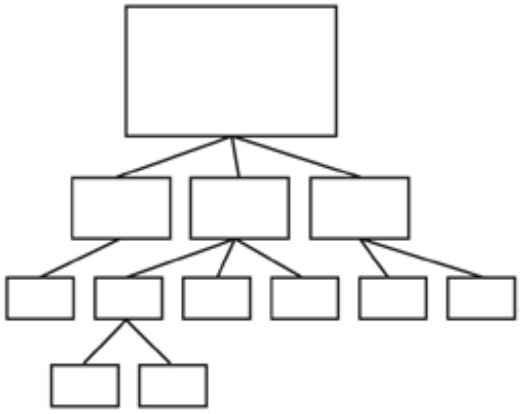
12 Rules Defined by Dr.Codd for a Relational Database Management System

10. **Integrity Independence- Database language should allow describing integrity rules, store these rules in online catalog using a common database language such as SQL.**
11. **Distribution Independence**
12. **Integrity rules should not be able to be violated from within any application. Integrity rules can be redefined using only a common database language such as SQL.**

"A Relational Model of Data for Large Shared Data Banks", 1970,Dr.E.F.Codd

Introduction to SQL

In many RDBMSs, there is a relation of parent/child relation.
Main nodes contain information about the locations of their children.



Summary of SQL

SQL is a universal language to manipulate and query data in RDBMS.

Using SQL, a Database Administrator (DBA) may

1. **Change the structure of Database in RDBMS**
2. **Administer the system security**
3. **Can give access to users for databases and objects in databases**
4. **Query a database for information**
5. **Manipulate data content of a database**
6. **Define transactions during manipulation of data**

Summary of SQL

SQL language contains 4 parts

1. **Data Definition Language**
2. **Data Manipulation Language**
3. **Data Query Language**
4. **Data Control Language**

Popular SQL DBMSs

1. **IBM DB2**
2. **IBM AS/400**
3. **ORACLE**
4. **MS SQL Server**
5. **InterBase**
6. **MYSQL**
7. **PostgreSQL**
8. **SYBASE**
9. **PROGRESS**
10. **INFORMIX**
11. **Ingres**
12. **Red Brick**
13. **SQLBase**
14. **Rdb**
15. **WATCOM/SQL Anywhere**
16. **Teradata**
17. **FoxPro**
18. **dBase**
19. **MS Access**
20. **Paradox**
21. **Clipper**

SQL Data Definition Language

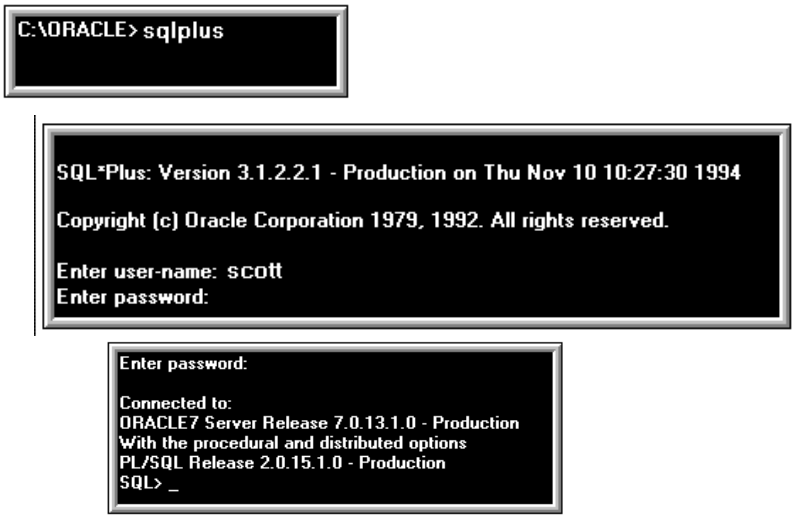
1. **Access to Database Management System**
2. **Data Types**
3. **Data Definition Language Statements**

cetinerg@itu.edu.tr

SQL Data Definition Language

1. Access to Database Management System

For Example; Using SQL*Plus under Oracle RDBMS

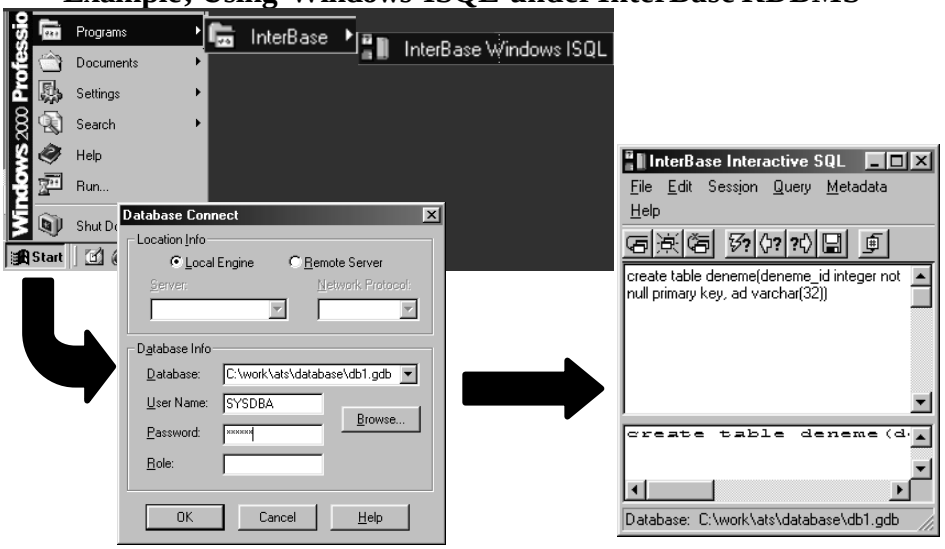


cetinerg@itu.edu.tr

SQL Data Definition Language

1. Access to Database Management System

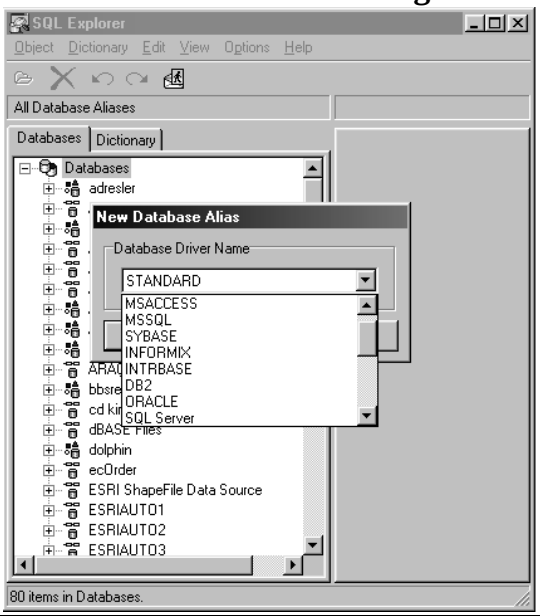
Example; Using Windows ISQL under InterBase RDBMS



cetinerg@itu.edu.tr

SQL Data Definition Language

1. Access to Database Management System



Example ; Using Generic Tools

Borland SQL Explorer can be connected to any supported database management system

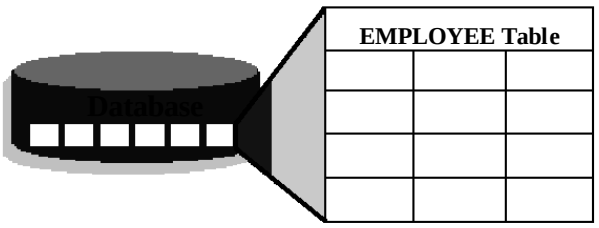
SQL Data Definition Language

2. DATA TYPES

Data are held in tables in Database Management System (DBMS)
The most important entity in a Relational Database is table.

SQL Data Definition Language

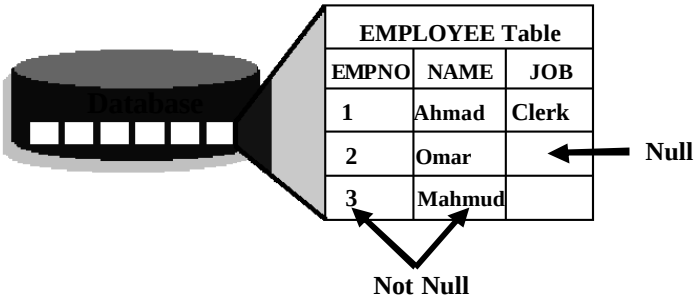
2. DATA TYPES



Data are held in tables in Database Management System (DBMS)
Tables consist of ROWS and COLUMNS.

SQL Data Definition Language

2. DATA TYPES



Data are held in tables in Database Management System (DBMS) Tables consist of ROWS and COLUMNS.

Each COLUMN has a DATA TYPE and may have a constraint not to have null values.

SQL Data Definition Language

2. DATA TYPES

Null Values

***Null* values are used to represent data for which we don't always have an applicable value. They represent optional attributes in data models. Think of the column values below which can take NULL values.**

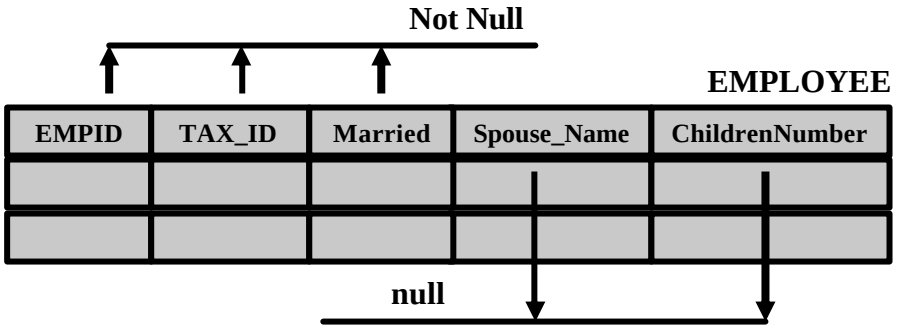
EMPLOYEE

EMPID	TAX_ID	Married	Spouse_Name	ChildrenNumber

For mandatory columns we have to have a constraint called NOT NULL.

2. DATA TYPES

Null Values



In EMPLOYEE Table, *Spouse_Name and ChildrenNumber* COLUMNS are not applicable for all EMPLOYEES. Therefore, they are nullable but other columns should take *NOT NULL* constraint.

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
MS SQL SERVER	BINARY BINARY() IMAGE VARBINARY VARBINARY()	DATETIME SMALLDATETIME TIMESTAMP	INT MONEY NUMERIC NUMERIC() NUMERIC(,) REAL SMALLINT SMALLMONEY TINYINT	CHAR CHAR() TEXT VARCHAR VARCHAR()
IBM DB2	GRAPHIC GRAPHIC() LONG VARGRAPHIC VARGRAPHIC()	DATE TIME TIMESTAMP	DECIMAL DECIMAL(,) FLOAT FLOAT() INTEGER REAL SMALLINT	CHAR CHAR() character character() LONG VARCHAR VARCHAR()

SQL Data Definition Language

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
ORACLE	LONG LONG RAW MLSLABEL RAW MLSLABEL RAW()	DATE	DECIMAL() DECIMAL(,) FLOAT INTEGER NUMBER NUMBER(*) NUMBER(,) SMALLINT	CHAR() character() LONG VARCHAR VARCHAR() VARCHAR2()
INTERBASE	BLOB	DATE	DECIMAL DECIMAL() DECIMAL(,) DOUBLE PRECISION FLOAT INTEGER NUMERIC NUMERIC() NUMERIC(,) SMALLINT	CHAR VARCHAR()

SQL Data Definition Language

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
INGRES	BYTE VARYING LONG BYTE	DATE	BYTE DECIMAL FLOAT FLOAT() FLOAT4 FLOAT8 INTEGER INTEGER1 INTEGER2 INTEGER4 MONEY SMALLINT	C CHAR() LONG VARCHAR TEXT() VARCHAR()
MS ACCESS	OLE OBJECT	DATE/TIME	AUTONUMBER BYTE CURRENCY DOUBLE INTEGER LONG INTEGER REPLICATION ID SINGLE YES/NO	MEMO TEXT()

cetinerg@itu.edu.tr *Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000*

SQL Data Definition Language

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
PARADOX	BINARY BINARY() GRAPHIC GRAPHIC() OLE OLE()	DATE TIME TIMESTAMP	AUTOINCREMENT BCD BCD() BYTES() LOGICAL LONG INTEGER MONEY NUMBER SHORT	ALPHA() FORMATTED MEMO FORMATTED MEMO() MEMO()
SYBASE	BINARY() IMAGE VARBINARY()	DATE/TIME SMALLDATETIME TIMESTAMP	BIT DECIMAL DECIMAL() DECIMAL(,) FLOAT INT MONEY NUMERIC NUMERIC() NUMERIC(,) REAL SMALLINT SMALLMONEY TINYINT	CHAR() TEXT VARCHAR()

SQL Data Definition Language

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
INFORMIX	BYTE	DATE	DEC DEC(,) DECIMAL DECIMAL(,) DOUBLE PRECISION DOUBLE PRECISION() FLOAT FLOAT() INT INTEGER MONEY MONEY(,) NUMERIC NUMERIC(,) REAL SERIAL SERIAL() SERIAL(1) SMALLFLOAT SMALLINT	CHAR CHAR() character character() NCHAR NCHAR() NVARCHAR() TEXT VARCHAR()

cetinerg@itu.edu.tr *Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000*

2. DATA TYPES

Data Base Name	BLOB DATA TYPE	DATE/TIME DATA TYPE	NUMBER DATA TYPE	STRING DATA TYPE
PROGRESS		DATE	DECIMAL DECIMAL() DECIMAL(,) FLOAT FLOAT() INTEGER LOGICAL NUMERIC NUMERIC() NUMERIC(,) REAL SMALLINT	CHAR() character()
FOXPRO ve DBASE IV	MEMO	DATE	FLOAT(,) LOGICAL NUMERIC(,)	character()

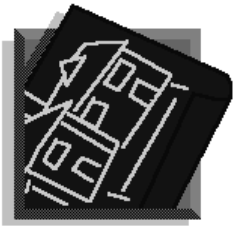
3. Data Definition Language Statements: CREATE TABLE

Table structure is defined according to specific data types available under a certain DBMS

3. Data Definition Language Statements: CREATE TABLE

There are 2 steps to construct a table.

1. DESIGN **2. DEFINITION**



3. Data Definition Language Statements: CREATE TABLE

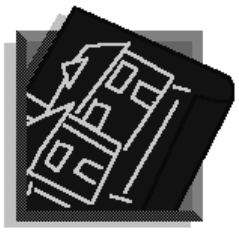
Steps in DESIGN Phase;

1. **Define Data Elements (Entities) to trace (Construct Data Model)**
2. **Define the Columns under Tables according to attributes of entities**
3. **Name the COLUMNS in Tables. Decide DATA TYPE in detail**
4. **Decide which COLUMN is *null* and which COLUMN has *not null* constraint. Mandatory attributes in data model become not null columns.**
5. **Define *Primary key Column(s)* and *Foreign key Column(s)***

3. Data Definition Language Statements: CREATE TABLE

Following steps are for definition Stage

1. **Create Table**
2. **Alter the Table for defining primary and/or foreign keys**



3. Data Definition Language Statements: CREATE TABLE

A SQL Statement to create a table must contain

- **Table Name**
- **COLUMN Name**
- **COLUMN DATA TYPE**

Now let's look at the rules to name tables;

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

1. **Depending on the DBMS, do not use more than some specific number of characters when naming the Table (For example; Table name should have a value of 1-30 character for ORACLE and 1-31 for INTERBASE respectively)**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

1. **Depending on the DBMS, do not use more than some specific number of characters when naming the Table (For example; Table name should have a value of 1-30 character for ORACLE and 1-31 for INTERBASE respectively)**
2. **The first character in Table name should be alphabetic one.**

cetinerg@itu.edu.tr *Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000*

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

1. **Depending on the DBMS, do not use more than some specific number of characters when naming the Table (For example; Table name should have a value of 1-30 character for ORACLE and 1-31 for INTERBASE respectively)**
2. **The first character in Table name should be alphabetic one.**
3. **Table name should consist of following characters. (Note: Although DBMSs like MS Access, Paradox allow using special characters it is better to avoid them for transferability)**
   - **A-Z, a-z (in English Alphabet), 0-9,_(underscore)**
   - **$ and # is valid for Oracle but not suggested**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

4. **Do not use a reserved word in DBMS. (For example; you cannot name a Table column as foreign)**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

4. **Do not use a reserved word in DBMS. (For example; you cannot name a Table name as Table)**
5. **You cannot use a word that you defined before (For example; if you defined a Table named Employees before, you cannot create a table with the same name)**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table (specific to Oracle but many applied to other DBMS)

4. **Do not use a reserved word in DBMS. (For example; you cannot name a Table column as foreign)**
5. **You cannot use a word that you defined before (For example; if you defined a Table named Employees before, you cannot create a table with the same name)**

Most DBMSs are not case-sensitive, i.e. EMPLOYEE and EmpLoYeE are same thing.

3. Data Definition Language Statements: CREATE TABLE

Which of the followings is not a valid table name?

a . CUSTOMER ORDERS

b . CUSTOMER_ORDERS

c . ORDERS

d . CUST_ORDERS

3. Data Definition Language Statements: CREATE TABLE

Now Rules for naming COLUMNS:

Table Name

COLUMN Name

COLUMN DATA TYPE

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table Column
(specific to Oracle but many applied to other DBMS)

1. **Depending the DBMS, the number of characters are limited for naming the COLUMNS. (For Example; this number is 1-30 for ORACLE and 1-31 for INTERBASE respectively.**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table Column
(specific to Oracle but many applied to other DBMS)

1. **Depending the DBMS, the number of characters are limited for naming the COLUMNS. (For Example; this number is 1-30 for ORACLE and 1-31 for INTERBASE respectively.**
2. **The first character in COLUMN Name should be alphabetic one**

cetinerg@itu.edu.tr *Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000*

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table Column
(specific to Oracle but many applied to other DBMS)

1. **Depending the DBMS, the number of characters are limited for naming the COLUMNS. (For Example; this number is 1-30 for ORACLE and 1-31 for INTERBASE respectively.**
2. **The first character in COLUMN Name should be alphabetic one**
3. **It should only contain following characters. (Some DBMSs such as MS Access, Paradox accept special characters but avoid using special characters for transferability.**

- **A-Z, a-z (in English Alphabet), 0-9,_(underscore)**
- **$ and # signs are valid for Oracle but avoid for transferability**

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table Column
(specific to Oracle but many applied to other DBMS)

4. **You cannot use certain reserved words in DBMS for naming the columns (For example; you cannot have a column named *foreign*)**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Rules for Naming a Table Column
(specific to Oracle but many applied to other DBMS)

4. **You cannot use certain reserved words in DBMS for naming the columns (For example; you cannot have a column named *foreign*)**
5. **Under the same table you cannot have two different columns with the same name. (You can have same column names under different tables)**

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

Tablo Name

COLUMN Name

COLUMN DATA TYPE

The last thing to do is to decide COLUMN DATA TYPES

cetinerg@itu.edu.tr *Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000*

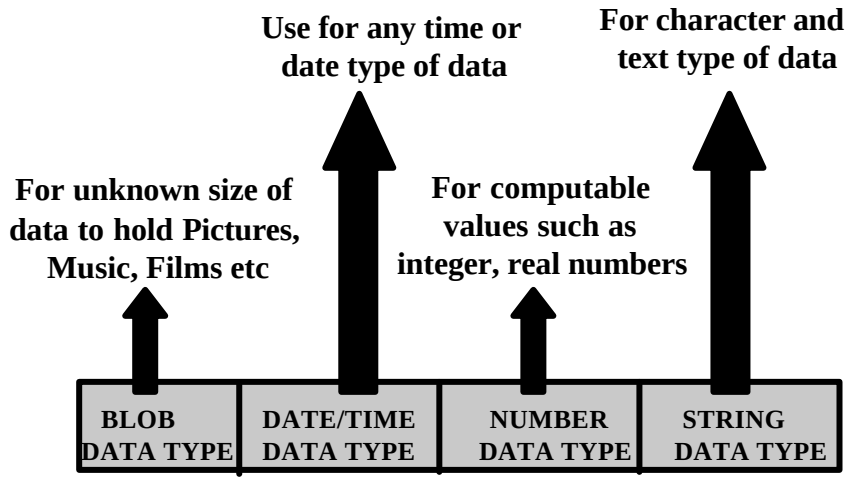
3. Data Definition Language Statements: CREATE TABLE

Be careful in defining COLUMN DATA TYPES.

For Example; Under ORACLE if you choose Number as data type, you can make calculations based on that column but not with data type called RAW.

3. Data Definition Language Statements: CREATE TABLE

USE OF DATA TYPES



3. Data Definition Language Statements: CREATE TABLE

USE OF DATA TYPES (EXAMPLE; ORACLE)

STRING DATA TYPE:`CHAR()` and `VARCHAR()`

For example; If you define a column as `CHAR(10)` ;

'Jeddah'
'United_Sta'
'UNITED_STA' are examples of data.

Queries are case sensitive for string data types, i.e. 'London' and 'LONDON' are two different things.

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

USE OF DATA TYPES (EXAMPLE; ORACLE)

STRING DATA TYPE:`CHAR()` and `VARCHAR()`

Difference between `VARCHAR(10)` and `CHAR(10)`

If you use CHAR(10) defining a column, it reserves a place of 10 characters in DBMS even if you insert less characters. If you enter value of 'Jeddah', DBMS stores it as 'Jeddah '.

VARCHAR(10) is more economical to use.

3. Data Definition Language Statements: CREATE TABLE

USE OF DATA TYPES (EXAMPLE; ORACLE)

BLOB DATA TYPE: `LONG`

This data type holds data up to 2 *gigabytes*. You can store text with more than 255 characters or pictures.
For each table, you can only have one LONG Data Type.

SQL Data Definition Language

3. Data Definition Language Statements: CREATE TABLE

USE OF DATA TYPES (EXAMPLE; ORACLE)

NUMBER DATA TYPE: `NUMBER(,)`

`NUMBER(,)` DATA TYPE is used to hold numbers. For example; By defining a column as `NUMBER(7,2)` you can hold `12345.67`. If you enter a value of 12345.678, it will be rounded to 12345.68.

3. Data Definition Language Statements: CREATE TABLE

Now you can use CREATE TABLE Statement. This statement consists of 5 things.

1. **Define Table Name**
2. **Define number and names of columns**
3. **Define data type for each column**
4. **Define for each column whether we allow null values or not.**
5. **Define other constraints for each column (primary key or foreign key constraints)**

3. Data Definition Language Statements: CREATE TABLE

CREATE TABLE *syntax is as follows:*

```
CREATE TABLE [User_Name.]table_name
(Column_Name Data_Type [column_constraint],
 Column_Name Data_Type [column_constraint],
              .....                  )
```

3. Data Definition Language Statements: CREATE TABLE

Following statement defines a table named EMPLOYEE .

```
CREATE TABLE EMPLOYEE
(EMP_ID              INTEGER     NOT NULL,
 NAME                VARCHAR(19) NOT NULL,
 SURNAME             VARCHAR(19) NOT NULL,
 IS_MARRIED          VARCHAR(1)  NOT NULL,
 NUMBER_OF_CHILDREN  SMALLINT    );
```

COLUMN NAMES **DATA TYPES** **NULL/NOT NULL**

3. Data Definition Language Statements: CREATE TABLE

Maximum number of columns is 255 for ORACLE DBMS. Usually we don't need that much if we have a proper relational design)

```
CREATE TABLE EMPLOYEE
(EMP_ID              INTEGER     NOT NULL,
 NAME                VARCHAR(19) NOT NULL,
 SURNAME             VARCHAR(19) NOT NULL,
 IS_MARRIED          VARCHAR(1)  NOT NULL,
 NUMBER_OF_CHILDREN  SMALLINT    );
```

COLUMN NAMES **DATA TYPES** **NULL/NOT NULL**

3. Data Definition Language Statements: CREATE TABLE

Second column is for data type of each column.

```
CREATE TABLE EMPLOYEE
(EMP_ID              INTEGER     NOT NULL,
 NAME                VARCHAR(19) NOT NULL,
 SURNAME             VARCHAR(19) NOT NULL,
 IS_MARRIED          VARCHAR(1)  NOT NULL,
 NUMBER_OF_CHILDREN  SMALLINT    );
```

COLUMN NAMES **DATA TYPES** **NULL/NOT NULL**

3. Data Definition Language Statements: CREATE TABLE

Third column shows whether the column is allowed to accept Null values. If the column is required then place NOT NULL.

```
CREATE TABLE EMPLOYEE
(EMP_ID              INTEGER     NOT NULL,
 NAME                VARCHAR(19) NOT NULL,
 SURNAME             VARCHAR(19) NOT NULL,
 IS_MARRIED          VARCHAR(1)  NOT NULL,
 NUMBER_OF_CHILDREN  SMALLINT    );
```

COLUMN NAMES **DATA TYPES** **NULL/NOT NULL**

3. Data Definition Language Statements: CREATE TABLE

What is the DATA TYPE for DEPT_ID?

```
CREATE TABLE DEPARTMENT
(DEPT_ID        INTEGER     NOT NULL,
 NAME           VARCHAR(32) NOT NULL,
 LOCATION       NUMBER(2),
 HEAD_DEPT_ID   INTEGER );
```

a. **VARCHAR(32)**
b. **NOT NULL**
c. **NUMBER(2)**
d. **INTEGER**

3. Data Definition Language Statements: CREATE TABLE

COLUMN CONSTRAINTS

You can place one or more constraint on the same column. In the following example, the column *DEPT_ID* takes two constraints (*not null* constraint and *primary key* constraint).

```
CREATE TABLE DEPARTMENT
(DEPT_ID    INTEGER NOT NULL CONSTRAINT pk_DEPT_ID PRIMARY KEY,
 NAME       VARCHAR(32) NOT NULL,
 LOCATION   NUMERIC(2));
```

3. Data Definition Language Statements: CREATE TABLE

COLUMN CONSTRAINTS

You can also write constraints at the end.

```
CREATE TABLE DEPARTMENT
(DEPT_ID    INTEGER NOT NULL CONSTRAINT pk_DEPT_ID PRIMARY KEY,
 NAME       VARCHAR(32) NOT NULL,
 LOCATION   NUMERIC(2));
```

```
CREATE TABLE DEPARTMENT
(DEPT_ID    INTEGER NOT NULL,
 NAME       VARCHAR(32) NOT NULL,
 LOCATION   NUMERIC(2),
 CONSTRAINT pk_DEPT_ID PRIMARY KEY);
```

Therefore, the statements above give the same result.

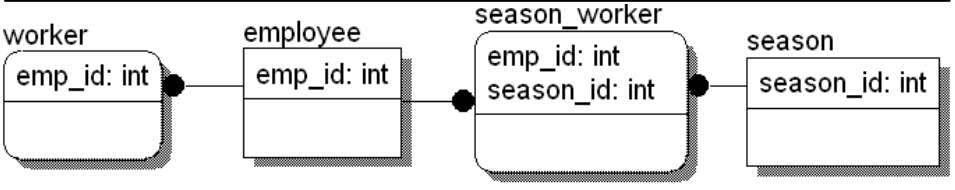
3. Data Definition Language Statements: EXAMPLE

```
create table employee (emp_no int not null constraint pk_employee primary key);

create table worker    (emp_no int not null constraint pk_worker primary key
                                constraint fk_worker references employee);

create table season(season_id int not null constraint pk_season primary key);

create table season_worker (emp_no int not null,
    season_id int not null,
    constraint pk_season_worker primary key (emp_no,season_id),
    constraint fk_season_worker1 foreign key(season_id) references season(season_id),
    constraint fk_season_worker2 foreign key(emp_no) references employee(emp_no)
                );
```



3. Data Definition Language Statements: CREATE TABLE

COPYING A TABLE UNDER ORACLE

If you have an existing table, you can create another table based on that one by using CREATE TABLE AS under ORACLE. This is not available in all DBMS.

Note: It does not copy primary key or foreign key constraints.

The syntax for CREATE TABLE AS is given below;

```
CREATE TABLE table_name[(COLUMN_Name,COLUMN_Name...)]
AS
  [Query]
```


3. Data Definition Language Statements: CREATE TABLE AS

Following CREATE TABLE AS statement creates a table called HDATES. New table contains three columns based on the query. They are EMPNO,ENAME and HIREDATE. New table includes three columns in all rows.

```
SQL> CREATE TABLE HDATES
2    AS
3         SELECT EMPNO,ENAME,HIREDATE
4         FROM EMP;
```

3. Data Definition Language Statements: CREATE TABLE AS

HDATES contains all data in EMP table.

```
SQL> CREATE TABLE HDATES
2    AS
3         SELECT *
4         FROM EMP;
```


3. Data Definition Language Statements: CREATE TABLE AS

What happens in the following example?

```
SQL> CREATE TABLE BLANK_EMP
2    AS
3         SELECT * FROM EMP
4         WHERE 2=3;
```

BLANK_EMP table is created with all columns as in EMP. But it contains no data from EMP because 2=3 condition is not met.

SQL Data Definition Language

3. Data Definition Language Statements: ALTER TABLE

After using CREATE TABLE statements, you can change the structure of your database by using ALTER TABLE ... Statements.

SQL Data Definition Language

3. Data Definition Language Statements: ALTER TABLE

Using ALTER TABLE ... statements you can make changes in table structures as follows:

- **Adding new a column or constraint. For example, adding a *NOT NULL* or *foreign key* constraint.**
- **Changing an existing column definition. You can change data type or constraint for an existing column.**

3. Data Definition Language Statements: ALTER TABLE

Using ALTER TABLE ... statements you can make changes in table structures as follows:

- **Adding new a column or constraint. For example, adding a *NOT NULL* or *foreign key* constraint.**
- **Changing an existing column definition. You can change data type or constraint for an existing column.**

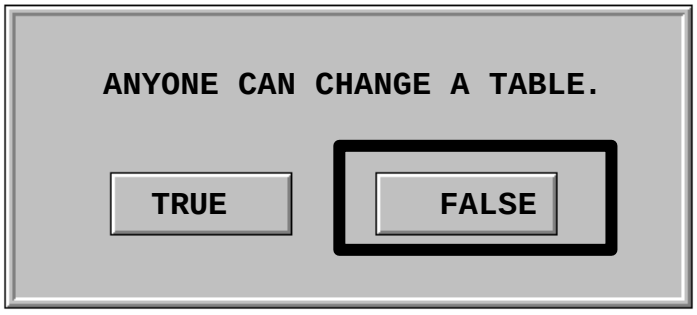
Who can change a Table structure?

Owner of the table or the persons allowed by the owner can change the structure of a table using ALTER TABLE statements.

Also DBA has got the privilege of access and change to any table.

3. Data Definition Language Statements: ALTER TABLE

True or False?



3. Data Definition Language Statements: ALTER TABLE

Adding a New COLUMN to a Table

The first change to a table is to add a column and constraint. For this purpose, you can use ALTER TABLE statement as follows:

```
ALTER TABLE Table_Name
  ADD ( {COLUMN_Name data_type | COLUMN_constraint},
       [{COLUMN_Name data_type | COLUMN_constraint}]..)
```

ORACLE

```
ALTER TABLE Table_Name
  ADD {COLUMN_Name data_type | COLUMN_constraint},
 [ADD {COLUMN_Name data_type | COLUMN_constraint}] ..
```

InterBase

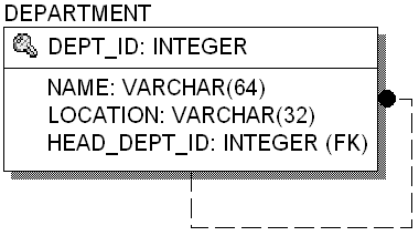
cetinerg@itu.edu.tr

3. Data Definition Language Statements: ALTER TABLE

By using ALTER TABLE ... Statement, you can change an existing column.

```
CREATE TABLE DEPARTMENT (
       DEPT_ID       INTEGER      NOT NULL PRIMARY KEY,
       NAME          VARCHAR(64)  NOT NULL,
       LOCATION      VARCHAR(32),
       HEAD_DEPT_ID  INTEGER      NOT NULL );
```

First construct a CREATE TABLE statement to convert the data model into design. The name of table is *DEPARTMENT*.



***HEAD_DEPT_ID* is not referenced yet as *foreign key*.**

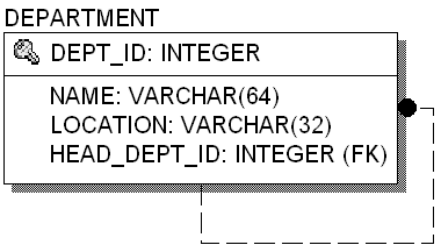
cetinerg@itu.edu.tr

3. Data Definition Language Statements: ALTER TABLE

ALTER TABLE Table_Name ADD FOREIGN KEY...

```
ALTER TABLE DEPARTMENT
       ADD FOREIGN KEY (HEAD_DEPT_ID)
                        REFERENCES DEPARTMENT;
```

For recursive relationship, *HEAD_DEPT_ID* is referenced from the same table.



3. Data Definition Language Statements: ALTER TABLE

ALTER TABLE Table_Name
ADD COLUMN_Name data_type constraint;

```
ALTER TABLE DEPARTMENT
       ADD NUMBER_OF_POSITIONS INTEGER;
```

We add a column named *NUMBER_OF_POSITIONS*

3. Data Definition Language Statements: ALTER TABLE

You can add many columns at the same time.
For example;

```
ALTER TABLE DEPARTMENT
       ADD NUMBER_OF_POSITIONS INTEGER,
       ADD MANAGER INTEGER NOT NULL;
```


3. Data Definition Language Statements: ALTER TABLE

CHANGING COLUMN DEFINITIONS

Using ALTER TABLE statement, you can change column structure and its constraints

Examples;

- **You can increase the width of a string column**
- **You can reduce the width of a string column**

3. Data Definition Language Statements: ALTER TABLE

Changing COLUMN Definitions

Syntax for changing COLUMN definitions;

```
ALTER TABLE Table_Name
 MODIFY ({COLUMN_Name data_type | COLUMN_constraint}
        [{COLUMN_Name data_type | COLUMN_constraint}] ..)
```

ORACLE

MODIFY is available under ORACLE. For most of the DBMSs, this choice is not available. For other DBMSs, you can first delete column using drop choice. And later use add choice to add column. Do not forget to get a backup of column before deleting.

3. Data Definition Language Statements: ALTER TABLE

ALTER TABLE Table_Name DROP COLUMN_Name

```
ALTER TABLE DEPARTMENT DROP NUMBER_OF_POSITIONS;
```

InterBase

Drop choice is used to delete a column in a table (ALTER TABLE ... DROP statement)

3. Data Definition Language Statements: DROP TABLE

DROP TABLE Statement

DROP TABLE statement deletes a table from the database permanently.

3. Data Definition Language Statements: DROP TABLE

DROP TABLE Statement

Syntax is very simple but the result is

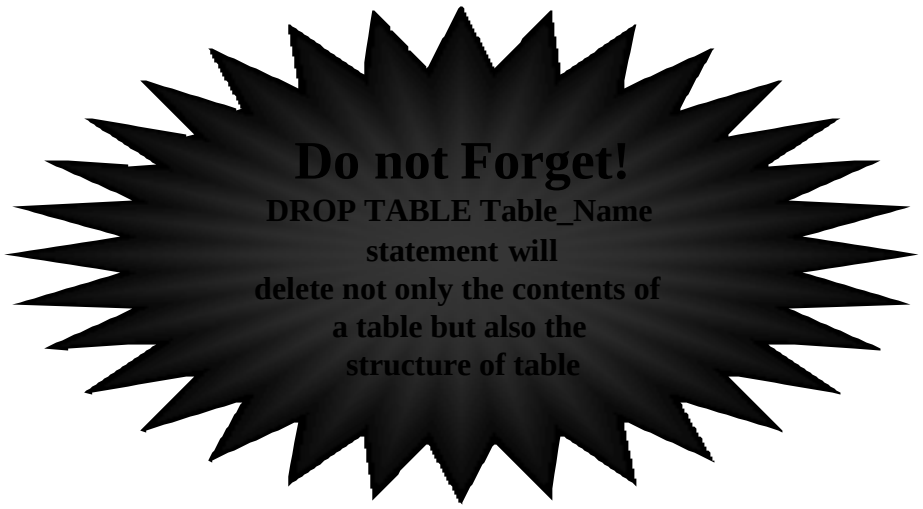
```
DROP TABLE Table_Name
```

Following statement deletes the table EMPLOYEE from database permanently.

```
DROP TABLE EMPLOYEE
```

SQL Data Definition Language

3. Data Definition Language Statements: DROP TABLE



cetinerg@itu.edu.tr

SQL Data Definition Language

3. Data Definition Language Statements

OTHER DATA DEFINITION LANGUAGE STATEMENTS

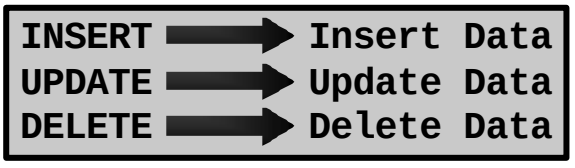
DEFINITON STATEMENTS	RESTRUCTURING STATEMENTS	DROP STATEMENTS
CREATE DATABASE CREATE DOMAIN CREATE EXCEPTION CREATE GENERATOR CREATE INDEX CREATE PROCEDURE CREATE ROLE CREATE SHADOW CREATE TABLE CREATE TRIGGER CREATE VIEW	ALTER DATABASE ALTER DOMAIN ALTER EXCEPTION ALTER INDEX ALTER PROCEDURE ALTER TABLE ALTER TRIGGER	DROP DATABASE DROP DOMAIN DROP EXCEPTION DROP EXTERNAL DROP FILTER DROP INDEX DROP PROCEDURE DROP ROLE DROP SHADOW DROP TABLE DROP TRIGGER DROP VIEW

NOTE: Some statements above are for INTERBASE DBMS

cetinerg@itu.edu.tr

SQL Data Manipulation Language

SQL Data Manipulation Language



SQL Data Manipulation Language

INSERT → Insert Data

INSERT statement is used to enter new rows into tables.

You have to know the structure of table before inserting data. Some DBMSs have some commands to learn the structure of a table, e.g. *DESCRIBE Table_Name* under ORACLE.

You can learn which columns exist in the table by using *SELECT * FROM Table_Name* statement. This query does not give column constraints.

SQL Data Manipulation Language

INSERT → Insert Data

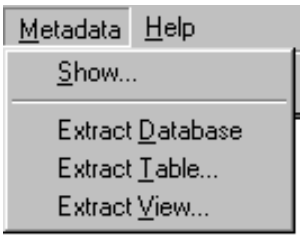
Example; Under ORACLE you can use *DESCRIBE NATION* statement to show the contents of *NATION* table.

```
SQL> DESCRIBE nation
Name          Null?        Type
------------- ------------ ------------
NATCODE       NOT NULL     NUMBER(5)
NATION        NOT NULL     VARCHAR2(28)
CAPITAL                    VARCHAR2(10)
AREA                       NUMBER(22)
POPULATION                 NUMBER(22)
```

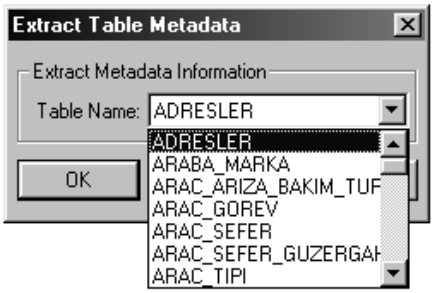
- ***Name*** **column shows the columns in table. NATION Table consists of NTCODE, NATION, CAPITAL, AREA and POPULATION columns.**
- ***Null?*** **column shows whether the column will take null values.**
- ***Type*** **column shows data type and size of the column.**

SQL Data Manipulation Language

INSERT → Insert Data



Using Windows ISQL under InterBase DBMS, you can learn table structure by *Metadata/Extract Table* choice.

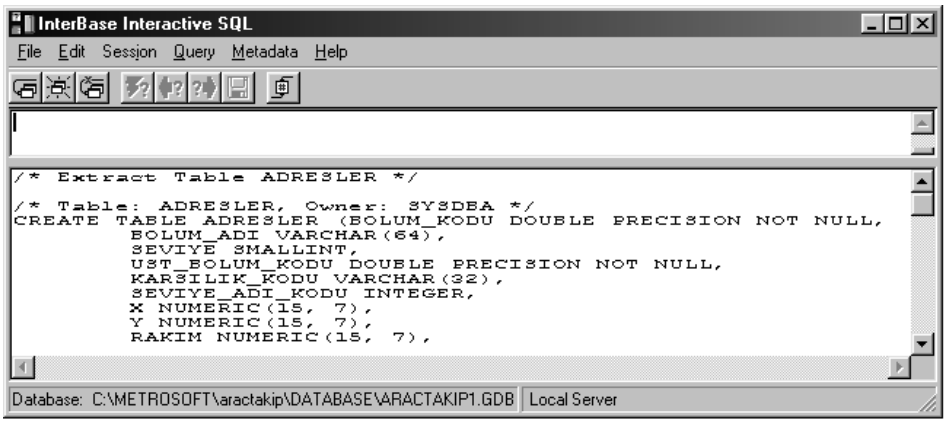


Suppose you have chosen a table named ADRESLER …

SQL Data Manipulation Language

INSERT → Insert Data

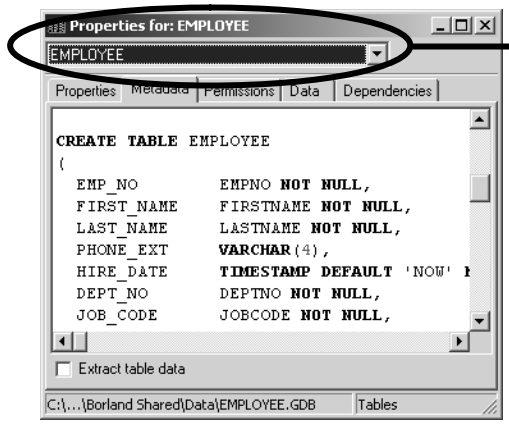
The result is shown as CREATE TABLE ..... Statement .



cetinerg@itu.edu.tr

INSERT → Insert Data

In new version of InterBase DBMS, go to Table properties and choose Metadata page



EMPLOYEE Table

INSERT → Insert Data

Now you know which table has which columns, data types and constraints.

The syntax for INSERT statement is below.

```
INSERT INTO Table_Name [(COLUMN1,COLUMN2...)]
                       VALUES (value1,value2...)
```

SQL Data Manipulation Language

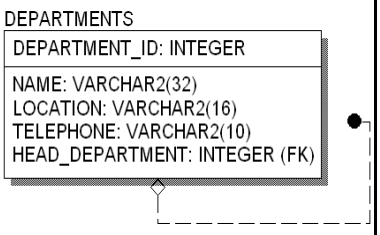
INSERT → Insert Data

INSERT statement can be used in 3 ways:

- **To insert a row with all column values**
- **To insert a row by specifying some columns**
- **To insert row (or rows) by querying another tables using SELECT statement**

SQL Data Manipulation Language

INSERT → Insert Data



INSERT statement can be used in 3 ways:

- **To insert a row with all column values**

Suppose you have a table of DEPARTMENTS and you want to specify all columns as follows.

DEPARTMENT_ID=12 NAME='Marketing and Sales'
LOCATION='High Street' TELEPHONE='285 31 95'
HEAD_DEPARTMENT=1

Required INSERT statement;

```
INSERT INTO DEPARTMENTS
  VALUES (12,'Marketing and Sales','High Street','285 31 95',1 )
```

SQL Data Manipulation Language

INSERT → Insert Data

Columns in INSERT statement should be specified according to the order in CREATE TABLE statement.

```
INSERT INTO DEPARTMENTS
  VALUES (12,'Marketing and Sales','High Street','285 31 95',1 )
```

NO COLUMN NAME

```
CREATE TABLE DEPARTMENTS (
      DEPARTMENT_ID        INTEGER NOT NULL,
      NAME                 VARCHAR2(32) NOT NULL,
      LOCATION             VARCHAR2(16) NULL,
      TELEPHONE            VARCHAR2(10) NULL,
      HEAD_DEPARTMENT      INTEGER NULL,
      PRIMARY KEY (DEPARTMENT_ID),
      FOREIGN KEY (HEAD_DEPARTMENT)
                            REFERENCES DEPARTMENTS
);
```


SQL Data Manipulation Language

INSERT → Insert Data

- **To insert a row by specifying some columns**
  You have to specify at least the not null columns. Column order may be mixed.

```
INSERT INTO DEPARTMENTS (NAME,DEPARTMENT_ID)
  VALUES ( 'Marketing and Sales', 12);
```

```
CREATE TABLE DEPARTMENTS (
      DEPARTMENT_ID        INTEGER NOT NULL,
      NAME                 VARCHAR2(32) NOT NULL,
      LOCATION             VARCHAR2(16) NULL,
      TELEPHONE            VARCHAR2(10) NULL,
      HEAD_DEPARTMENT      INTEGER NULL,
      PRIMARY KEY (DEPARTMENT_ID),
      FOREIGN KEY (HEAD_DEPARTMENT) REFERENCES DEPARTMENTS);
```

SQL Data Manipulation Language

INSERT → Insert Data

To insert row (or rows) by querying another tables using SELECT statement

SQL Data Manipulation Language

Using INSERT and SELECT together

Syntax is as follows to insert row (or rows) by querying another tables using SELECT statement

```
INSERT INTO TableToBeInserted [(COLUMN1,COLUMN2...)]
  SELECT COLUMN1,COLUMN2... FROM TableToBeSelected
         [WHERE .....]
```

NOTE: Column data types and order in INSERT part should be compatible in the second (SELECT) part.

Using INSERT and SELECT together

First, let us define a table called DEPARTMENTS_BACKUP by following statement.

```
CREATE TABLE DEPARTMENTS_BACKUP (DEPARTMENT INTEGER NOT NULL,
                                 NAME       VARCHAR(64))
```


Using INSERT and SELECT together

Following statement copies the rows and columns of table DEPARTMENT for departments starting with 'A' into DEPARTMENTS_BACKUP Table.

```
INSERT INTO DEPARTMENTS(DEPARTMENT , NAME)
                SELECT DEPARTMENT_ID , NAME
                 FROM DEPARTMENTS WHERE NAME LIKE 'A%'
```

COLUMN names may be different but data types should be same.

SQL Data Manipulation Language

cetinerg@itu.edu.tr

SQL Data Manipulation Language

UPDATE statements are used to change data in columns.

UPDATE Statement may be used

1. To update columns in one row

2. To update columns in many rows

cetinerg@itu.edu.tr

UPDATE → Update Data

UPDATING COLUMNS OF A ROW

Suppose you want to update DEPARTMENTS Table as follows;

for DEPARTMENT_ID=19

```
DEPARTMENTS Table

COLUMN Name      Null?    DATA TYPE
------------   -------  ----------------
DEPARTMENT_ID   NOT NULL    INTEGER
NAME            NOT NULL    VARCHAR(64)
HEAD_DEPARTMENT             INTEGER
```

NEW VALUES
NAME='Quality Department' HEAD_DEPARTMENT=1

Two columns will be updated herein.

UPDATE → Update Data

Syntax for UPDATE Statement

```
UPDATE <Table_Name>
    SET <COLUMN1> = <value1> [, <COLUMN2> = <value2> .... ]
    [WHERE condition]
```

There are 2 parts in UPDATE Statement as follows;

- **SET clause is used to specify column names to be updated and new values for these columns.**
- **WHERE clause is optional. Defines which rows should be changed. Where statement must be a condition which produces TRUE or FALSE.**

SQL Data Manipulation Language

UPDATE → Update Data

Syntax for UPDATE Statement

```
UPDATE <Table_Name>
    SET <COLUMN1> = <value1> [, <COLUMN2> = <value2> .... ]
    [WHERE condition]
```

What is update statement to update the values of NAME and HEAD_DEPARTMENT in DEPARTMENTS Table with the following values for DEPARTMENT_ID=19 ?

NEW VALUES
NAME='Quality Control' HEAD_DEPARTMENT=1

```
UPDATE DEPARTMENTS
    SET NAME='Quality Control', HEAD_DEPARTMENT=1
    WHERE DEPARTMENT_ID=19;
```


SQL Data Manipulation Language

UPDATE → Update Data

```
UPDATE DEPARTMENTS
    SET NAME='Quality Control', HEAD_DEPARTMENT=1
    WHERE DEPARTMENT_ID=19;
```

Before UPDATE

DEPARTMENT_ID	NAME	HEAD_DEPARTMENT
19	Quality Assurance Department	2

After UPDATE

DEPARTMENT_ID	NAME	HEAD_DEPARTMENT
19	Quality Control	1

Answer the following question.

You want to change the values as VEHICLE_TYPE='SERVICE' and PLATE='34-UN-1600' in VEHICLES Table.

VEHICLE_ID	VEHICLE_TYPE	PLATE	MODEL	STATE
7	TOYOTA SLX	34-CD-123	1999	HIRED

Will the following UPDATE Statement perform this change?

```
UPDATE VEHICLES
   SET VEHICLE_TYPE='SERVICE', PLATE='34-UN-1600';
```

TRUE **FALSE**

NOTE: Since there is no WHERE condition, all rows will be changed. It is dangerous to use UPDATE Statement without condition.

Updating many rows

Using UPDATE statement, you can change many rows at the same time. For example; you want to make 20% increase in salaries of employees who are 'PROGRAMMER'.

Using UPDATE statement as in the next slide, you can make this change.

Updating Multiple Rows

Make 20% increase in salaries of employees who are 'PROGRAMMER'.

```
UPDATE EMPLOYEE
    SET NET_SALARY=NET_SALARY+NET_SALARY*0,20
       WHERE JOB='PROGRAMMER';
```

You can display changes using following SELECT Statement .

```
SELECT EMP_ID,SURNAME,JOB,NET_SALARY
 FROM EMPLOYEE;
```


You can display changes using following SELECT Statement.

Before UPDATE

EMP_ID	SURNAME	JOB	NET_SALARY
4250	ÖZER	PROGRAMMER	10000
4351	ÖZTÜRK	DIRECTOR	12500
4432	SELVI	PROGRAMMER	9000
3321	SEROGLU	OPERATOR	3500

After UPDATE

EMP_ID	SURNAME	JOB	NET_SALARY
4250	ÖZER	PROGRAMMER	12000
4351	ÖZTÜRK	DIRECTOR	12500
4432	SELVI	PROGRAMMER	10800
3321	SEROGLU	OPERATOR	3500

In EMPLOYEE Table (for EMP_ID=4251), you want to change the value of DEPARTMENT column as 'Marketing/Sales'.

What is the UPDATE statement for this?

A
```
UPDATE EMPLOYEE
    SET DEPARTMENT= 'Marketing/Sales'
       WHERE DEPARTMENT_ID=4251;
```

B
```
UPDATE EMPLOYEE
    SET EMP_ID=4251
       WHERE DEPARTMENT= 'Marketing/Sales'
```

C
```
UPDATE EMPLOYEE
    SET DEPARTMENT= 'Marketing/Sales'
       WHERE EMP_ID=4251;
```

D
```
UPDATE EMPLOYEE
    SET DEPARTMENT= 'Marketing/Sales';
```


Update Data with NULL Values

For columns which can take null values, you can update some columns as NULL values.

Example

```
UPDATE EMPLOYEE
   SET base_salary=base_salary*1.20,department=NULL
   WHERE JOB='Salesman'
       AND HIREDATE<'1-JAN-1992'
```

SQL Data Manipulation Language

Update Data using Queries

You can supply search condition in WHERE clause as Queries. For example, you want to update the manager of all employees as 123 working in the same department with 'Khalid Omar'. The first thing to do is to learn the *department_id* of 'Khalid Omar'. Use the following SELECT Statement for this;

```
SELECT department FROM EMPLOYEE
   WHERE name='Khalid' AND surname='Omar'
```

Suppose we get the value of 13 for *department_id*. Now you can update EMPLOYEE table with the following statement;

```
UPDATE EMPLOYEE
  SET manager=123
       WHERE department_id=13
```


SQL Data Manipulation Language

Update Data using Queries

You can join UPDATE and SELECT Statements for the same purpose.

```
UPDATE EMPLOYEE
  SET manager=123
       WHERE department_id=
          (SELECT department FROM EMPLOYEE
             WHERE name='Khalid' AND surname='Omar')
```

Deleting Rows from Tables

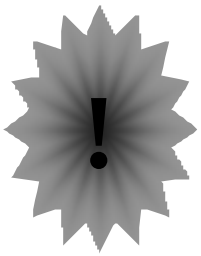
Using SQL Delete statements you can delete rows from tables

The syntax for DELETE Statement is given below.

```
DELETE FROM Table_Name
    [WHERE condition]
```

Suppose *Employee* with surname 'ESEN' left the job.
You want to delete his record under *EMPLOYEE* table.
You can do this with following statement;

```
DELETE FROM EMPLOYEE
WHERE SURNAME='ESEN';
```



What happens if there are more than one employees with surname 'ESEN'?

```
DELETE FROM EMPLOYEE
WHERE SURNAME='ESEN';
```

All employees with surname 'ESEN' will be deleted.

It is suggested that you use a primary key value in where condition not to delete any record by mistake.

```
DELETE FROM EMPLOYEE
     WHERE EMP_ID=4251;
```

```
DELETE FROM EMPLOYEE;
```

What happens after executing the SQL statement above?

A. Nothing. Statement is not executed due to error.

B. An error message is given.

C. EMPLOYEE Table is removed completely.

D. All rows in EMPLOYEE Table are deleted.

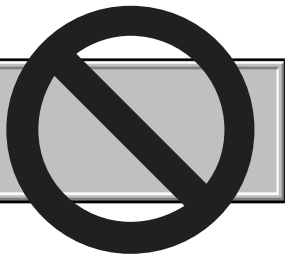
SQL Data Manipulation Language

DELETE → Delete Data

Be careful with using DELETE Statement.

It will delete all values in table.

```
DELETE FROM EMPLOYEE;
```



cetinerg@itu.edu.tr

Assoc.Prof.Dr.B.Gültekin Çetiner ? 2000

SQL Data Transaction Language

Transaction Concept
COMMIT Statement
ROLLBACK Statement
SAVEPOINT Statement (Not available in all DBMSs)

SQL Data Transaction Language

What is a Transaction?

Let us try to explain *Transaction* with one example.
You went to ACM (Automated Cash Machine) to get some money. System updates your balance after subtracting the money you have taken. Suppose something bad happened (electricity cut) and you couldn't get your money at the end.

Will it be subtracted from your account?

We define all processes as *one transaction* until they really finish and made permanent in database.

A transaction is one single process containing chained processes.

We should be able to get back all changes whenever necessary.

SQL Data Transaction Language

What is a Transaction?

Technically, Transaction is a group of SQL statement defined as one unit.

A transaction may for example contain following SQL statements;

- **INSERT Statement**
- **Another INSERT Statement**
- **DELETE Statement**
- **UPDATE Statement**

The changes made will be permanent by using COMMIT Statement. If you want to cancel a transaction then you can use ROLLBACK Statement. Transactions vary according to the job you are performing.

SQL Data Transaction Language

COMMIT Statement

To make a transaction permanent you should use COMMIT Statement at the end of transaction. After COMMIT, all processes within transaction group are made permanent. A typical transaction is given below;

Transaction

1. **Insert a new employee row into EMPLOYEE Table**
2. **Update Salary of a specified EMPLOYEE**
3. **Update department for an employee**
4. **Delete the row of record for one employee**

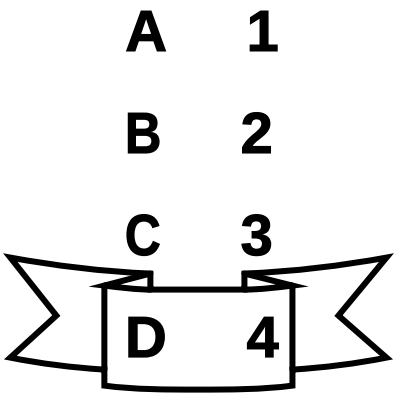
```
COMMIT;
```

COMMIT Statement makes these 4 SQL Statements permanent in database. Since there is no other COMMIT Statement, these 4 SQL Statements are named as only one transaction.

SQL Data Transaction Language

There are some SQL statements below. How many transactions are available?

```
INSERT .....
UPDATE .....
DELETE .....
COMMIT;
INSERT .....
DELETE .....
COMMIT;
UPDATE .....
INSERT .....
COMMIT;
UPDATE .....
COMMIT;
```

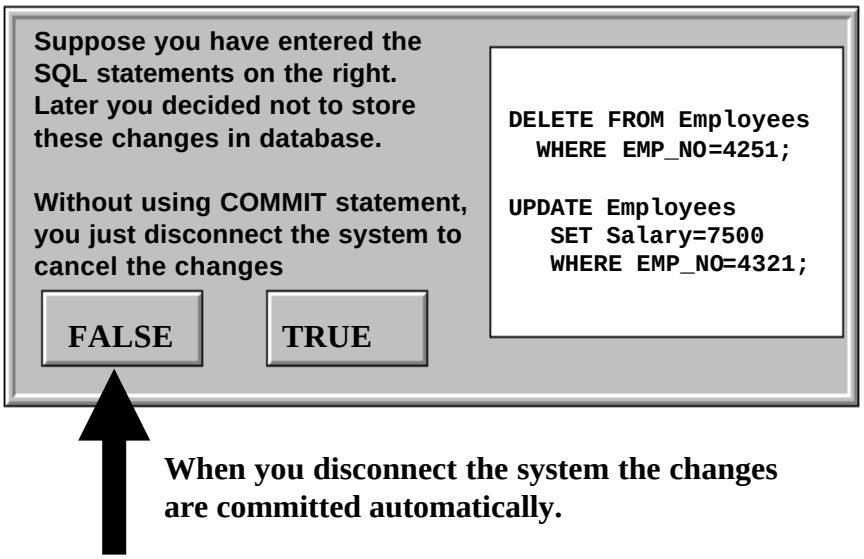


SQL Data Transaction Language

Transactions are temporary until they are committed. You can see the changes you have made through INSERT, UPDATE and DELETE statements but other users cannot see the changes you have made until you or system COMMIT these changes.

You have to use COMMIT statement to make your transactions permanent.

If you disconnect the system. your changes will be automatically committed by the DBMS.



When you disconnect the system the changes are committed automatically.

ROLLBACK Statement

Using ROLLBACK Statement, you can cancel all changes in transaction.

SQL Data Transaction Language

ROLLBACK Statement

Let's see how we execute ROLLBACK Statements. Following SQL Statements shows there is one transaction which is committed and one group uncommitted.

```
INSERT .....
UPDATE .....
DELETE .....
COMMIT;
INSERT .....
DELETE .....
UPDATE .....
```


SQL Data Transaction Language

ROLLBACK Statement

If you want to cancel the last 3 SQL Statements then you use ROLLBACK.

```
INSERT .....
UPDATE .....
DELETE .....
COMMIT;
```

ROLLBACK Statement

If you want to cancel the last 3 SQL Statements then you use ROLLBACK.

```
INSERT .....
UPDATE .....
DELETE .....
COMMIT;
```

All changes are cancelled after last COMMIT Statement. INSERT, UPDATE and DELETE statements cannot be cancelled since they are COMMITTED. You can rollback up to last commit statement.

First Salary value is 5000 for employee with *emp_id=4251.* What is Salary value after SQL statements?

- 5000
- 8500
- 9000
- Hiçbiri

```
UPDATE Employee
  SET Salary=8500
  WHERE emp_no=4251;
COMMIT;
UPDATE Employee
  SET Salary=9000
  WHERE emp_no=4251;
COMMIT;
ROLLBACK;
```

COMMIT is after ROLLBACK Statement.
There is nothing to cancel.

SAVEPOINT Statement (Under ORACLE DBMS)

ROLLBACK Statement will cancel all changes after the last COMMIT Statement. In some DBMS such as ORACLE, there are alternatives to cancel part of the transaction with ROLLBACK.

You can use SAVEPOINT Statement under Oracle DBMS. SAVEPOINT is a point to which you can rollback the changes.

The syntax for SAVEPOINT is

```
SAVEPOINT Savepoint_name;
```

***Savepoint_name* herein is a variable name. You can cancel all changes up to this point.**

Following pages show an example for using *Savepoints*.

SQL Data Transaction Language

The following transaction starts process by inserting a new line in a table.
Later, this transaction is made permanent in Database by COMMIT Statement.
A new transaction starts with UPDATE statement.

```
INSERT .....
COMMIT;
UPDATE .....
```


SQL Data Transaction Language

Let's define a SAVEPOINT in a transaction as in the following example.
After SAVEPOINT statement there is DELETE statement. This transaction contains one UPDATE and one DELETE statement (INSERT is another transaction which is finished by COMMIT).
Suppose you want to cancel only the result of DELETE.

```
INSERT .....
COMMIT;
UPDATE .....
SAVEPOINT FirstPoint;
DELETE .....
```

SQL Data Transaction Language

Here, you can use ROLLBACK statement. The syntax for ROLLBACK statement is given below;

```
ROLLBACK
    [TO [SAVEPOINT] ReturnPoint];
```

Using TO SAVEPOINT parameter you can cancel only part of transaction (cancel after the ReturnPoint).

SQL Data Transaction Language

The following ROLLBACK statement will only cancel the result of DELETE statement.

```
INSERT .....
COMMIT;
UPDATE .....
SAVEPOINT firstpoint;
DELETE .....
ROLLBACK TO SAVEPOINT firstpoint;
```

Why SAVEPOINT?

In a Transaction, you can define certain points using SAVEPOINT. Transactions may sometimes contain several time consuming functions/processes. You can define SAVEPOINTs before executing functions. If there is error in any point, you can turn back to any SAVEPOINT. In that case you don't need to re-execute the functions before this SAVEPOINT. You can continue the transaction after this point.

```
Define Savepoint 1
A Time Consuming Function
Define Savepoint 2
Another Time Consuming Process
      ERROR
Go to  Savepoint 2.....
```

cetinerg@itu.edu.tr

For the Employee 4251, initial Salary value is 5000.
What is the new Salary value after applying the SQL statements on the right?

- 500
- **8500**
- 9000
- None

```
UPDATE Employees
  SET Salary=8500
  WHERE EMP_NO=4251;
SAVEPOINT point1;
UPDATE Employees
  SET Salary=9000
  WHERE EMP_NO=4251;
ROLLBACK TO
  SAVEPOINT point1;
COMMIT;
```

cetinerg@itu.edu.tr

SQL Data Query Language

Data from Databases are retrieved using SELECT statements.

SELECT statements are the richest part of SQL language.

Since they were the subjects taught before this course they will not be discussed herein.

cetinerg@itu.edu.tr