BCA313

Unit 4

SQL

Dr Arpita Mathur
Assistant Professor, Faculty of Computer Science,
LMCST, Jodhpur

History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- And as we will see later, also other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Domain Types in SQL

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric**(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 4.

Create Table Construct

- An SQL relation is defined using the **create table** command:

  create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2))

Integrity Constraints in Create Table

- **not null**
- **Unique key <column name> <data type>(<size>) UNIQUE**
  - At table level **UNIQUE (<Column name1>,<Column name2>….)**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

Example:

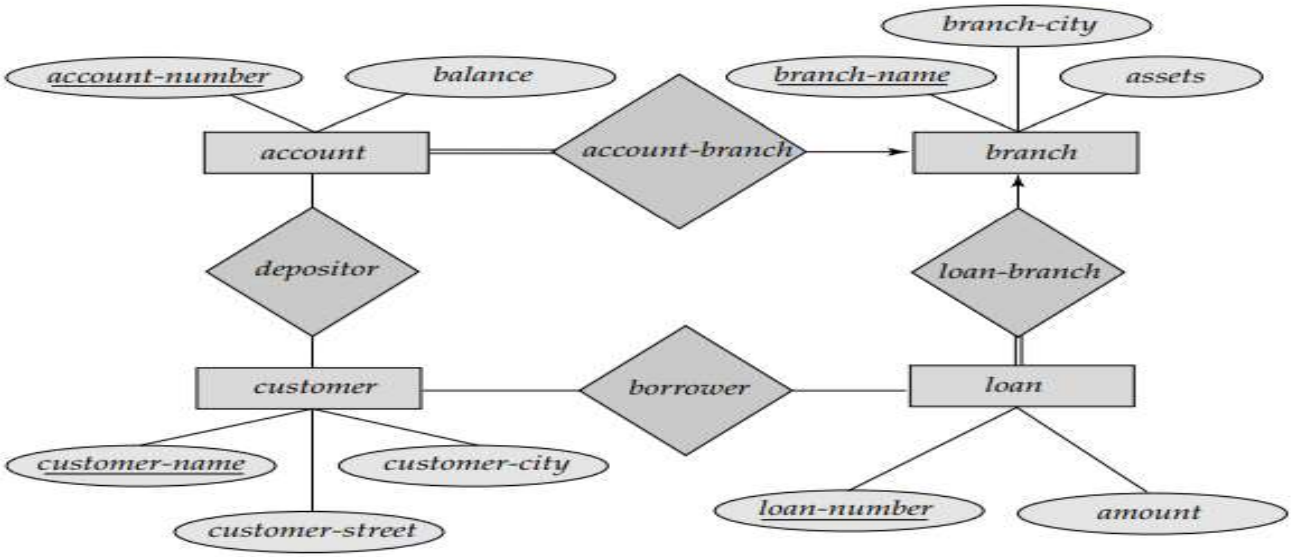
create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department);*

primary key declaration on an attribute automatically ensures **not null**

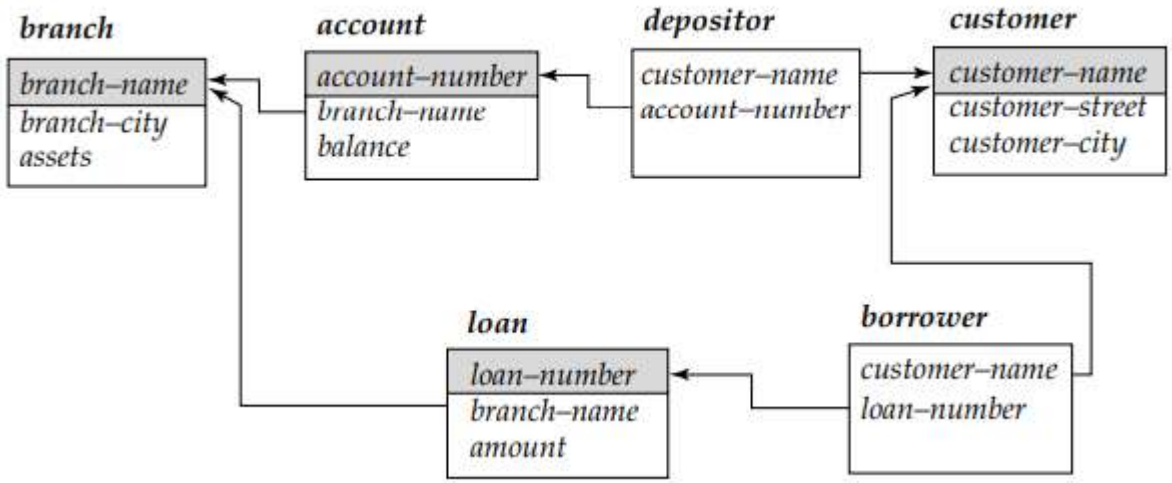
CONSTRAINT <constraint name> <constraint defination>

Eg: CONSTRAINT f_deptkey **FOREIGN KEY** *(dept_name*) **references** *department*

E R diagram for Banking Enterprise



Schema diagram for Banking Enterprise



And a Few More Relation Definitions

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key *(ID),*
  foreign key *(dept_name*) **references** *department*);

- **create table** *takes* (
  ID **varchar**(5),
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  primary key *(ID, course_id, sec_id, semester, year)* ,
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);
  - Note: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

And more still

- **create table** *course* (
  course_id **varchar**(8),
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  primary key *(course_id),*
  foreign key *(dept_name*) **references** *department*);

Updates to tables

- **Insert**
  - **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);
- **Delete**
  - Remove all tuples from the *student* relation
    - **delete from** *student*
- **Drop Table**
  - **drop table** *r*
- **Alter**
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

MODIFYING THE STRUCTURE OF TABLES

The structure of a table can be modified by using the **ALTER TABLE** command. **ALTER TABLE** allows **changing the structure of an existing table**. With **ALTER TABLE** it is possible to **add** or **delete** columns, **create** or **destroy** indexes, **change the data type** of existing columns, or **rename columns** or the **table** itself.

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. While **ALTER TABLE** is executing, the original table is still readable by users of Oracle.

Updates and **writes** to the table are **stalled** until the new table is ready, and then are automatically redirected to the new table **without any failed updates**.

Note

To use **ALTER TABLE**, the **ALTER**, **INSERT**, and **CREATE** privileges for the table are required.

Adding New Columns

Syntax:

```
ALTER TABLE <TableName>
    ADD(<NewColumnName> <Datatype> (<Size>),
        <NewColumnName> <Datatype> (<Size>)...);
```

Example 21:
Enter a new field called City in the table BRANCH_MSTR.

```
ALTER TABLE BRANCH_MSTR ADD (CITY VARCHAR2(25));
```

Output:

```
Table altered.
```

Dropping A Column From A Table

Syntax:

```
ALTER TABLE <TableName> DROP COLUMN <ColumnName>;
```

Example 22:
Drop the column city from the BRANCH_MSTR table.

```
ALTER TABLE BRANCH_MSTR DROP COLUMN CITY;
```

Output:

```
Table altered.
```

Modifying Existing Columns

Syntax:

```
ALTER TABLE <TableName>
        MODIFY (<ColumnName> <NewDatatype>(<NewSize>));
```

Example 23:
Alter the BRANCH_MSTR table to allow the NAME field to hold maximum of 30 characters

```
ALTER TABLE BRANCH_MSTR MODIFY (NAME varchar2(30));
```

Output:

```
Table altered.
```

Restrictions on the ALTER TABLE

The following tasks **cannot** be performed when using the **ALTER TABLE** clause:

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

RENAMING TABLES

Oracle allows renaming of tables. The rename operation is done **atomically**, which means that **no other thread** can access any of the tables while the rename process is running.

Note

To rename a table the ALTER and DROP privileges on the original table, and the CREATE and INSERT privileges on the new table are required.

To rename a table, the syntax is

Syntax:

```
RENAME <TableName> TO <NewTableName>
```

Example 24:
Change the name of branches table to branch table

```
RENAME BRANCH_MSTR TO BRANCHES;
```

Output:

```
Table renamed.
```

TRUNCATING TABLES

TRUNCATE TABLE empties a table completely. Logically, this is equivalent to a **DELETE** statement that deletes all rows, but there are practical differences under some circumstances.

TRUNCATE TABLE differs from **DELETE** in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one
- Truncate operations are not transaction-safe (i.e. an error will occur if an active transaction or an active table lock exists)
- The number of deleted rows are not returned

Syntax:

```
TRUNCATE TABLE <TableName>;
```

Example 25:
Truncate the table BRANCH_MSTR

```
TRUNCATE TABLE BRANCH_MSTR;
```

Output:

```
Table truncated.
```

DESTROYING TABLES

Sometimes tables within a particular database become obsolete and need to be discarded. In such situation using the DROP TABLE statement with the table name can destroy a specific table.

Syntax:

```
DROP TABLE <TableName>;
```

Caution

If a table is dropped all records held within it are lost and cannot be recovered.

Example 26:
Remove the table BRANCH_MSTR along with the data held.

```
DROP TABLE BRANCH_MSTR;
```

Output:

```
Table dropped.
```

The following examples show the definitions of several integrity constraints:

Example 17:
Alter the table EMP_MSTR by adding a primary key on the column EMP_NO.

ALTER TABLE EMP_MSTR **ADD PRIMARY KEY** (EMP_NO);

Output:
```
Table altered.
```

Example 18:
Add FOREIGN KEY constraint on the column **VERI_EMP_NO** belonging to the table **FD_MSTR**, which references the table **EMP_MSTR**. Modify column **MANAGER_SIGN** to include the **NOT NULL** constraint

ALTER TABLE FD_MSTR **ADD CONSTRAINT** F_EmpKey **FOREIGN KEY**(VERI_EMP_NO)
REFERENCES EMP_MSTR **MODIFY**(MANAGER_SIGN **NOT NULL**);

Output:
```
Table altered.
```

DROPPING INTEGRITY CONSTRAINTS VIA THE ALTER TABLE COMMAND

Integrity constraint can be dropped if the rule that it enforces is no longer **true** or if the constraint is no longer **needed**. Drop the constraint using the **ALTER TABLE** command with the **DROP** clause. The following examples illustrate the dropping of integrity constraints:

Example 19:
Drop the PRIMARY KEY constraint from EMP_MSTR.

ALTER TABLE EMP_MSTR **DROP PRIMARY KEY;**

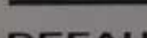
Output:
```
Table altered.
```

Example 20:
Drop FOREIGN KEY constraint on column VERI_EMP_NO in table FD_MSTR

ALTER TABLE FD_MSTR **DROP CONSTRAINT F_EmpKey;**

Output:
```
Table altered.
```

Note



Dropping UNIQUE and PRIMARY KEY constraints **also drops all** associated indexes.

DEFAULT VALUE CONCEPTS

At the time of table creation a **default value** can be assigned to a column. When a record is loaded into the table, and the column is left empty, the Oracle engine will automatically load this column with the default value specified. The data type of the default value should match the data type of the column. The **DEFAULT** clause can be used to specify a default value for a column.

Syntax:

<ColumnName> <Datatype>(<Size>) DEFAULT <Value>;

Example 21:

Create ACCT_MSTR table where the column CURBAL is the number and by default it should be zero. The other column STATUS is a varchar2 and by default it should have character **A**. (Refer to table definitions in the chapter 6)

```
CREATE TABLE "DBA_BANKSYS"."ACCT_MSTR"("ACCT_NO" VARCHAR2(10),
    "SF_NO" VARCHAR2(10), "LF_NO" VARCHAR2(10), "BRANCH_NO" VARCHAR2(10),
    "INTRO_CUST_NO" VARCHAR2(10), "INTRO_ACCT_NO" VARCHAR2(10),
    "INTRO_SIGN" VARCHAR2(1), "TYPE" VARCHAR2(2), "OPR_MODE" VARCHAR2(2),
    "CUR_ACCT_TYPE" VARCHAR2(4), "TITLE" VARCHAR2(30),
    "CORP_CUST_NO" VARCHAR2(10), "APLNDT" DATE, "OPNDT" DATE,
    "VERI_EMP_NO" VARCHAR2(10), "VERI_SIGN" VARCHAR2(1),
    "MANAGER_SIGN" VARCHAR2(1), "CURBAL" NUMBER(8, 2) DEFAULT 0,
    "STATUS" VARCHAR2(1) DEFAULT 'A');
```

Output:

```
Table created.
```

Note

- The data type of the default value should match the data type of the column
- Character and date values will be specified in single quotes
- If a column level constraint is defined on the column with a default value, the default value clause must precede the constraint definition

Thus the syntax will be:

<ColumnName> <Datatype>(<Size>) DEFAULT <Value> <constraint definition>

Basic Query Structure

- A typical SQL query has the form:

  select A_1, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.

The select Clause

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*
  from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select**.**
- Find the department names of all instructors, and remove duplicates

  select distinct *dept_name*
  from *instructor*

- The keyword **all** specifies that duplicates should not be removed.

  select all *dept_name*
  from *instructor*

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

  select *
  from *instructor*

- An attribute can be a literal with no **from** clause

  select ‘437’

  - Results is a table with one column and a single row with value “437”
  - Can give the column a name using:

    select ‘437’ **as** *FOO*

- An attribute can be a literal with **from** clause

  select ‘A’
  from *instructor*

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value “A”

The select Clause (Cont.)

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
  - The query:

    select *ID, name, salary/12*
    from *instructor*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.
  - Can rename "s*alary/12"* using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary*

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

  select *
  from *instructor*

- An attribute can be a literal with no **from** clause

  select ‘437’

  - Results is a table with one column and a single row with value “437”
  - Can give the column a name using:

    select ‘437’ **as** *FOO*

- An attribute can be a literal with **from** clause

  select ‘A’
  from *instructor*

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value “A”

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'
- Comparison results can be combined using the logical connectives **and, or,** and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

    select *name*
    from *instructor*
    where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000
- Comparisons can be applied to results of arithmetic expressions.