SQL
“Structured Query Language”

- Standard for relational db systems
- History:
  Developed at IBM in late 70s
  First standard: SQL-86
  Second standard: SQL-92
  Third standard: SQL-99 or SQL3, well over 1000 pages!
  “The nice things about standards is that you have so many to choose from” -- Andres S. Tannenbaum

SQL Data Definition Language

Allows the specification of the database schema: a set of relations with information about each relation

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- 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.
### Some 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.
- **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.

### Create Table Command

- An SQL relation is defined using the `create table` command:
  ```sql
  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 domain of attribute \( A_i \)

- Example:
  ```sql
  create table branch
  (branch_name char(15) not null,
  branch_city char(30),
  assets integer)
  ```
Create Table (cont.)

- Can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE keywords

```
CREATE TABLE DEPT
(  DNAME VARCHAR(10) NOT NULL,  
  DNUMBER INTEGER NOT NULL,  
  MGRSSN CHAR(9),  
  MGRSTARTDATE CHAR(9),  
  PRIMARY KEY (DNUMBER),  
  UNIQUE (DNAME),  
  FOREIGN KEY (MGRSSN) REFERENCES EMP );
```

- *primary key* declaration on an attribute automatically ensures *not null* in SQL-92 onwards, needs to be explicitly stated in SQL-89

Drop Table Command

- Used to remove a relation *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

```
DROP TABLE DEPENDENT;
```
Alter Table Command

• The **alter table** command is used to add attributes to an existing relation:
  \[
  \text{alter table } r \text{ add } A \ D
  \]
  where \( A \) is the name of the attribute to be added to relation \( r \) and \( D \) is the domain of \( A \).
  All tuples in the relation are assigned *null* as the default value for the new attribute.

• The **alter table** command can also be used to drop attributes of a relation:
  \[
  \text{alter table } r \text{ drop } A
  \]
  where \( A \) is the name of an attribute of relation \( r \)
  Many databases do *not* support dropping of attributes

Alter Table (cont.)

• Since new attribute will have NULL values right after the **ALTER** command is executed, the NOT NULL constraint is *not allowed* for such an attribute.

• Example:
  \[
  \text{ALTER TABLE EMPLOYEE}
  \text{ADD JOB VARCHAR(12);}
  \]

• The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the **UPDATE** command.
Integrity Constraints

• Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  – A savings account must have a balance greater than $10,000.00
  – A salary of a bank employee must be at least $6.00 an hour
  – A customer must have a (non-null) phone number

Constraints on Single Relations

• not null
• primary key
• unique
• check \(P\), where \(P\) is a predicate
The check clause

• **check** \((P)\), where \(P\) is a predicate

Declare `branch_name` as the primary key for `branch` and ensure that the values of `assets` are non-negative.

```sql
create table branch
  (branch_name char(15),
   branch_city char(30),
   assets integer,
   primary key (branch_name),
   CHECK (assets >= 0)
)
```

The check clause (Cont.)

• The **check** clause permits domains to be restricted:
  - Use **check** clause to ensure that an hourly_wage domain allows only values greater than a specified value.
    ```sql
    create domain hourly_wage numeric (5,2)
    constraint value_test check (value >= 4.00)
    ```
  - The domain has a constraint that ensures that the hourly_wage is greater than 4.00
  - The clause **constraint value_test** is optional; useful to indicate which constraint an update violated.
Referential Integrity

• Ensures that a value that appears in one relation for a given set of attributes also appears for a set of attributes in another relation.
  – Example: If “La Jolla” is a branch name appearing in one of the tuples in the account relation, then there exists a tuple in the branch relation for branch “La Jolla”.

• Foreign keys can be specified as part of the create table statement:
  – The foreign key clause lists foreign key (FK) attributes and the name of the relation referenced by the FK. By default, a FK references PK attributes of the referenced table.

Referential Integrity in SQL – Example

```sql
create table customer
  (customer_name    char(20),
   customer_street  char(30),
   customer_city    char(30),
   primary key      (customer_name ))
create table branch
  (branch_name      char(15),
   branch_city      char(30),
   assets           numeric(12,2),
   primary key      (branch_name ))
```
create table account
  (account_number char(10),
   branch_name char(15),
   balance integer,
   primary key (account_number),
   foreign key (branch_name) references branch )
create table depositor
  (customer_name char(20),
   account_number char(10),
   primary key (customer_name, account_number),
   foreign key (account_number) references account,
   foreign key (customer_name) references customer )