



Advanced SQL

Database System Concepts, 5th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use





Advanced SQL

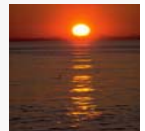
- SQL Data Types and Schemas
- Integrity Constraints
- Authorization
- Transaction





Built-in Data Types in SQL

- **date:** Dates, containing a (4 digit) year, month and date
 - Example: **date** '2005-7-27'
- **time:** Time of day, in hours, minutes and seconds.
 - Example: **time** '09:00:30' **time** '09:00:30.75'
- **timestamp:** date plus time of day
 - Example: **timestamp** '2005-7-27 09:00:30.75'
- **interval:** period of time
 - Example: **interval** '1' day
 - Subtracting a date/time/timestamp value from another gives an interval value
 - Interval values can be added to date/time/timestamp values

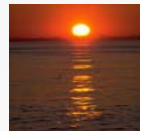




Build-in Data Types in SQL (Cont.)

- Can extract values of individual fields from date/time/timestamp
 - Example: **extract (year from r.starttime)**

- Can cast string types to date/time/timestamp
 - Example: **cast <string-valued-expression> as date**
 - Example: **cast <string-valued-expression> as time**





User-Defined Types

- **create type** construct in SQL creates user-defined type

```
create type Dollars as numeric (12,2) final
```

- **create domain** construct in SQL-92 creates user-defined domain types

```
create domain person_name char(20) not null
```

- Types and domains are similar. Domains can have constraints, such as **not null**, specified on them.





Domain Constraints

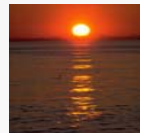
- **Domain constraints** are the most elementary form of integrity constraint. They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types
 - Example: **create domain Dollars numeric(12, 2)**
create domain Pounds numeric(12,2)
- We cannot assign or compare a value of type Dollars to a value of type Pounds.
 - However, we can convert type as below
(**cast r.A as Pounds**)
(Should also multiply by the dollar-to-pound conversion-rate)





Large-Object Types

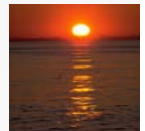
- Large objects (photos, videos, CAD files, etc.) are stored as a *large object*.
 - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
 - **clob**: character large object -- object is a large collection of character data
 - When a query returns a large object, a pointer is returned rather than the large object itself.





Integrity Constraints

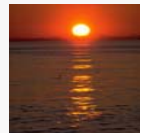
- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
 - A checking account must have a balance greater than \$10,000.00
 - A salary of a bank employee must be at least \$4.00 an hour
 - A customer must have a (non-null) phone number





Constraints on a Single Relation

- not null
- primary key
- unique
- check (P), where P is a predicate

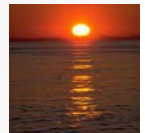




Not Null Constraint

- Declare *branch_name* for *branch* is **not null**
branch_name **char(15) not null**
- Declare the domain *Dollars* to be **not null**

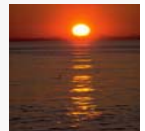
create domain *Dollars* numeric(12,2) not null





The Unique Constraint

- **unique** (A_1, A_2, \dots, A_m)
- The unique specification states that the attributes
 A_1, A_2, \dots, A_m
form a candidate key.
- Candidate keys are permitted to be null (in contrast to primary keys).





The check clause

- **check** (P), where P is a predicate

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

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



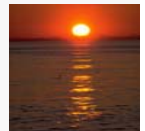


The check clause (Cont.)

- The **check** clause in SQL-92 permits domains to be restricted:
 - Use **check** clause to ensure that an `hourly_wage` domain allows only values greater than a specified value.

```
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 certain set of attributes in another relation.
 - Example: If “Perryridge” 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 “Perryridge”.
- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
 - The **primary key** clause lists attributes that comprise the primary key.
 - The **unique key** clause lists attributes that comprise a candidate key.
 - The **foreign key** clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key. By default, a foreign key references the primary key attributes of the referenced table.





Referential Integrity in SQL – Example

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 ))
```





Referential Integrity in SQL – Example (Cont.)

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*)





Assertions

- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form
create assertion <assertion-name> **check** <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
 - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Asserting
for all X , $P(X)$
is achieved in a round-about fashion using
not exists X such that not $P(X)$

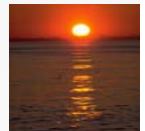




Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance or \$1000.00

```
create assertion balance_constraint check  
(not exists (  
  select *  
  
  from loan  
  
  where not exists (  
    select *  
    from borrower, depositor, account  
    where loan.loan_number = borrower.loan_number  
      and borrower.customer_name = depositor.customer_name  
      and depositor.account_number = account.account_number  
      and account.balance >= 1000)))
```

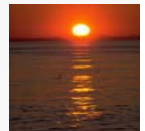




Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum_constraint check  
  (not exists (select *  
    from branch  
    where (select sum(amount)  
      from loan  
        where loan.branch_name =  
          branch.branch_name)  
    >= (select sum (amount)  
      from account  
        where loan.branch_name =  
          branch.branch_name )))
```





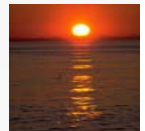
Authorization

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema (covered in Chapter 8):

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.





Authorization Specification in SQL

- The **grant** statement is used to confer authorization
 - grant** <privilege list>
 - on** <relation name or view name> **to** <user list>
- <user list> is:
 - a user-id
 - **public**, which allows all valid users the privilege granted
 - A role (more on this in Chapter 8)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).



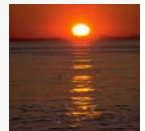


Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
 - Example: grant users U_1 , U_2 , and U_3 **select** authorization on the *branch* relation:

grant select on *branch* to U_1, U_2, U_3

- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges
- more in Chapter 8





Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.
revoke <privilege list>
on <relation name or view name> **from** <user list>
- Example:
revoke select on branch from U_1, U_2, U_3
- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public**, all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.





Transaction

- Transaction
 - Logical unit of access to database
- Transaction demarcation
 - Manual
 - ▶ Transaction boundary is specified explicitly
 - Beginning of a transaction
 - » Transaction begins when any kind of valid SQL command is issued or after the previous commit or rollback command
 - End of a transaction
 - » COMMIT : Commit all updates after the beginning of current transaction
 - » ROLLBACK : Cancel all updates
 - Automatic
 - ▶ Each SQL command is treated as a transaction
 - ▶ Command
 - SET AUTOCOMMIT ON | OFF





End of Chapter

Database System Concepts, 5th Ed.

©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use

