COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 9
STRUCTURED QUERY LANGUAGE (SQL)

EXAMPLE BANK RELATIONAL SCHEMA

Branch(branchName, branchCity, assets)

Client(clientName, clientStreet, clientCity)

Loan(loanNo, amount, *branchName*)

Account(accountNo, balance, *branchName*)

Borrower(*clientName, loanNo*)

Depositor(*clientName, accountNo*)

Attribute names in italics are foreign key attributes.

DATA DEFINITION LANGUAGE (DDL)

The SQL DDL allows the specification of:

- The schema for each relation (attributes).
- The types of values associated with each attribute (i.e., the domain of values the attribute, such as string, number, date, etc.).
- User-defined types and domains.
- Integrity constraints (ICs).
  - domain, key, foreign key, general
- The physical storage structure of each relation on disk.
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.

BASIC TYPES

char(*n*)	Fixed length character string with length *n*.
varchar2(*n*)	Variable-length character string with maximum length *n*.
int	An integer (a finite subset of the integers that is machine-dependent).
smallint	A small integer (a machine-dependent subset of the integer domain type).
number(*p*,*d*)	A fixed point number with a total of *p* digits (the precision) and *d* digits to the right of the decimal point.
float(*n*)	Floating point number, with user-specified precision of at least *n* digits.
date	A date containing a (4 digit) year, month and day of month.
time	The time of day, in hours, minutes and seconds.
timestamp	A combination of date and time.

- Null values are allowed in all the domain types.
- Declaring an attribute to be not null prohibits null values for that attribute.

☞ **Some relational systems also allow user-defined types.**

USER-DEFINED TYPES

- The create type clause is used to define a new type.

```
create type PersonName as char(25) final;
```

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

```
create type Pounds as numeric(12,2) final;
```

The keyword final is required by the SQL:1999 standard, but has no real meaning in this context; some implementations allow the final keyword to be omitted.

☞ **Not all relational systems support user-defined types.**

USER-DEFINED DOMAINS

- The create domain clause is used to define a new domain.

```
create domain hourlyWage numeric(5,2);
```

- Differences between user-defined types and domains:
  - Domains can have constraints, such as not null, specified on them and can have default values defined for variables of the domain; types cannot.
  - Domains are not strongly typed; types are strongly typed.

☞ **Not all relational systems support user-defined domains**.

CREATING RELATIONS

- The create table command is used to define and create a relation.

- The domain type of each attribute needs to be specified.
  - **Basic domain types:** char(*n*), varchar2(*n*), int, smallint, number(*p*,*d*), float(*n*), date, time, timestamp
  - A default value can be specified for an attribute.
  - Null values are allowed in all the basic domain types.

☞ **The domain type of an attribute is enforced by the DBMS whenever tuples are added or modified.**

```
create table Student (
   studentId    char(8) not null,
   name         varchar2(45) not null,
   email        varchar2(30),
   birthdate    date not null,
   cga          number(3,2));
```

```
create table EnrollsIn (
   studentId    char(8) not null,
   courseId     char(8) not null,
   grade        number(4,1) default 0 not null);
```

ALTERING AND DESTROYING RELATIONS

- The alter table command is used to add attributes to, modify attributes in or drop attributes from an existing relation.

Example:

```
alter table Student
    add firstYear int;
```

The schema is altered by adding a new attribute and extending every tuple in the current instance with a null value for the new attribute.

```
alter table Student
    drop column firstYear;
```

The schema is altered by dropping the attribute from the relation and deleting its value in every tuple.

- The drop table command deletes *all* information about a relation (both data *and* schema).

Example:

```
drop table Student;
```

INTEGRITY CONSTRAINTS (IC)

An integrity constraint (IC) **ensures that authorized changes to the database do not result in a loss of data consistency.**

☞ **An IC guards against accidental damage to the database.**

- ICs are based upon the requirements of the real-world application that is being described in the database relations.
  - An IC is a statement about *all possible* instances!
  - For the **Student** relation, we know, from common knowledge, that name is not a key, but the constraint that an attribute, such as studentId, is a key has to be given to us by the client.
- We can check a database instance to see if an IC is violated, but we can **never** infer that an IC is true by looking at a database instance. **Why?**

DOMAIN CONSTRAINTS

- Domain constraints define valid values for attributes.
- They test values inserted into the database and test queries to ensure that the comparisons make sense.
- Besides a basic domain type, additional constraints can be specified on attributes in the create table command.

not null	specifies that null values are not allowed
primary key	specifies a key for a relation (the value of a key attribute *cannot be null* $\Longrightarrow$ no need to specify not null)
unique	specifies that an attribute or a set of attributes is a candidate key (the attribute value(s) *can be null*)
foreign key	specifies that one or more attributes refer to a primary key attribute in another relation
check	specifies a predicate that the values in every tuple must satisfy

PRIMARY KEY, UNIQUE CONSTRAINTS

- A relation can possibly have many *candidate keys* one of which is chosen as the *primary key*.

```
create table Student (
    studentId    char(8),
    name         varchar2(45) not null,
    email        varchar2(30) not null,
    birthdate    date not null,
    cga          number(3,2),
    primary key (studentId),
    unique (email));
```

```
create table EnrollsIn (
    studentId    char(8),
    courseId     char(8) not null,
    grade        number(4,1) default 0 not null,
    primary key (studentId),
    unique (courseId, grade));
```

X

Used carelessly, an IC can prevent the storage of database instances that should be allowed!

FOREIGN KEY CONSTRAINT

Recall: A foreign key is a set of attributes in one relation whose values must match the values of the primary key in another relation or be null.

☞ **A foreign key must reference the primary key of the referenced relation.**

Example: Only students listed in the Student relation should be allowed to enroll for courses.

```
create table EnrollsIn (
    studentId    char(8),
    courseId     char(8),
    grade        number(4,1) default 0 not null,
    primary key (studentId, courseId),
    foreign key (studentId) references Student(studentId));
```

☞ **Every** studentId **value in the** EnrollsIn **relation must reference a tuple in the Student relation with a matching** studentId **value.**

FOREIGN KEY:
ENFORCING REFERENTIAL INTEGRITY

- What should be done if an EnrollsIn tuple with a non-existent student id is inserted?

☞ **Reject it!**

- What should be done if a Student tuple is deleted?

1. Disallow deletion of a Student tuple that is referred to by an EnrollsIn tuple (*default action*).
2. Alternatively, delete all EnrollsIn tuples that refer to it (on delete cascade).
3. Set studentId in EnrollsIn tuples that refer to it to a *default value* (on delete set default).
4. Set studentId in EnrollsIn tuples that refer to it to a *null value* (on delete set null).

☞ 3 and 4 are not applicable in the example since studentId is part of the primary key.

FOREIGN KEY: ENFORCING REFERENTIAL INTEGRITY (CONT'D)

- What should be done if the primary key student id of a tuple in Student is updated?

☞ **Reject it!**

- Alternatively, propagate the update to the tuples in the EnrollsIn relation with matching student ids (on update cascade).

```
create table EnrollsIn (
    studentId    char(8),
    courseId     char(10),
    grade        number(4,1) default 0 not null,
    primary key (studentId, courseId),
    foreign key (studentId) references Student(studentId)
        on delete cascade
        on update cascade);
```

The referential integrity actions in the referencing relation (EnrollsIn) are triggered when a tuple in the referenced relation (Student) is deleted or updated.

Oracle Note
Oracle does not support on update cascade.

CHECK CLAUSE: ATTRIBUTES

- The check clause is used to add an integrity constraint for an attribute and can contain an arbitrary predicate.
  - ☞ **The predicates are similar to those allowed in a where clause.**

- It is specified in the definition of a relation and checked whenever there is an update to the relation.

Example: Ensure that semester can have only specified values and that year is between 2020 and 2024.

```
create table Section (
    courseId     char(8),
    sectionId    char(2),
    semester     char(6),
    year         char(4) check (year between '2020' and '2024'),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer')));
```

CHECK CLAUSE: DOMAINS

- The check clause can be used in a create domain clause to add an integrity constraint to the domain.

Example: Ensure that an hourlyWage domain allows only values greater than a specified value.

```
create domain hourlyWage numeric(5, 2)
    constraint wageTest check (value>=40.00);
```

– The new domain hourlyWage is declared to be a decimal number with 5 digits, 2 of which are after the decimal point.

– The domain has a constraint named wageTest that ensures that hourlyWage is greater than 40.00.

☞ **The constraint name is optional, but useful to indicate which constraint an update violated.**

VIEWS

- Views provide a way to hide certain data from certain users.
- The create view command creates a view.

  create view *view-name* as <*query expression*>

  where: <*query expression*> is any legal SQL query.

Example: Create a view from the Loan relation that hides the amount.

```
create view BranchLoan as
select loanNo, branchName
from Loan;
```

Loan

loanNo	amount	branch Name
L-170	30000	Central
L-155	170000	Tsimshatsui
L-230	40000	Central

BranchLoan

loanNo	branch Name
L-170	Central
L-155	Tsimshatsui
L-230	Central

- The drop view command deletes a view.

Example:

```
drop view BranchLoan;
```

VIEWS (CONT'D)

- A view can be queried the same as any other relation.

  Example: Find all loans in the Central branch.

```
select loanNo
from BranchLoan
where branchName='Central';
```

BranchLoan

loanNo	branch Name
L-170	Central
L-155	Tsimshatsui
L-230	Central

loanNo
L-170
L-230

- A user who has access to the BranchLoan view, but not the Loan relation, cannot see the amount.

- Some database systems allow view relations to be stored and kept up-to-date if the relations used to define the view change.

  ☞ **Such views are called materialized views.**

VIEWS (CONT'D)

- Not all views can allow update operations.
- A view is updateable (i.e., tuples can be inserted, updated and deleted) if *all* of the following conditions are satisfied by the query that defines the view.
  - The from clause has only one relation.
  - The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
  - Any attribute not listed in the select clause can be set to **null** (i.e., it does not have a not null constraint and is not part of a primary key.
  - The query does not have a group by or having clause.

☞ **The specific rules for view update are often system dependent.**

TUPLE DELETION

- The delete command deletes *zero or more tuples* from a relation.

  Example: Delete all account tuples at the Central branch.

```
delete from Account
where branchName='Central';
```

- Conceptually, deletion is done in two steps.

  1. Find the tuples to delete.

```
select * from Account
where branchName='Central';
```

  2. Delete the tuples found.

☞ **Deletion can be done *only if* no integrity constraints are violated!**

COMPLEX DELETION

- The where clause predicate in a delete statement can be as complex as in a select statement.

Example: Delete all accounts at the Mongkok branch.

```
delete from Depositor
where accountNo in (select accountNo
                    from Depositor natural join Account
                    where branchName= 'Mongkok');
```

☞ **Must also delete the accounts of these depositors!**

TUPLE INSERTION

- The insert command adds one or more tuples to a relation.

Example: Add a new Account tuple.

```
insert into Account values ('A-732', 1200 , 'Central' );
```

Example: Add a new tuple to Account with balance set to null.

```
insert into Account values ('A-733', null, 'Central');
```

☞ **The order of the values must match the order of the attributes in the relation.**

- Attribute names need to be *specified explicitly* for order-independent insertion.

```
insert into Account (accountNo, branchName, balance)
    values ('A-734', 'Central', 1200);
```

COMPLEX INSERTION

- Insertion values can be obtained from the result of a query.

Example: Create a $200 savings account for all loan clients of the Central branch. Let the loan number serve as the account number for the new savings account.

```
insert into Account
select loanNo, 200, branchName
from Loan
where branchName='Central';
```

```
insert into Depositor
select clientName, loanNo
from Loan natural join Borrower
where branchName='Central';
```

The order of the attributes in the select clause must match the order of the attributes in the table being inserted into.

Note: The keyword values is omitted when the values to insert are obtained from a select statement.

TUPLE UPDATE

- The **update** command is used to change a value in a tuple.

Example: Increase all accounts with balance over $10,000 by 6%; all other accounts receive 5%.

```
update Account
set balance=balance*1.06
where balance>10000;
```

```
update Account
set balance=balance*1.05
where balance<=10000;
```

☞ **Need two update statements!**

☞ **The order is important! Why?**

CONDITIONAL UPDATE: CASE STATEMENT

- This update can be specified more easily using the case statement.

Example: Increase all accounts with balances over $10,000 by 6%; all other accounts receive 5%.

```
update Account
set balance= case
                when balance<=10000 then balance*1.05
                else balance*1.06
             end;
```