

# SQL Data Definition Language

- Relational Data Definition
- Example Relational Schema
- SQL Data Definition Language
- Defining a Database Schema
- Data Integrity
- Another Example Schema
- Default Values
- Defining Keys
- Attribute Value Constraints
- Named Constraints

## ❖ Relational Data Definition

In order to give a relational data model, we need to:

- describe tables
- describe attributes that comprise tables
- describe any constraints on the data

A relation schema defines an individual table

- table name, attribute names, attribute domains, keys, etc.

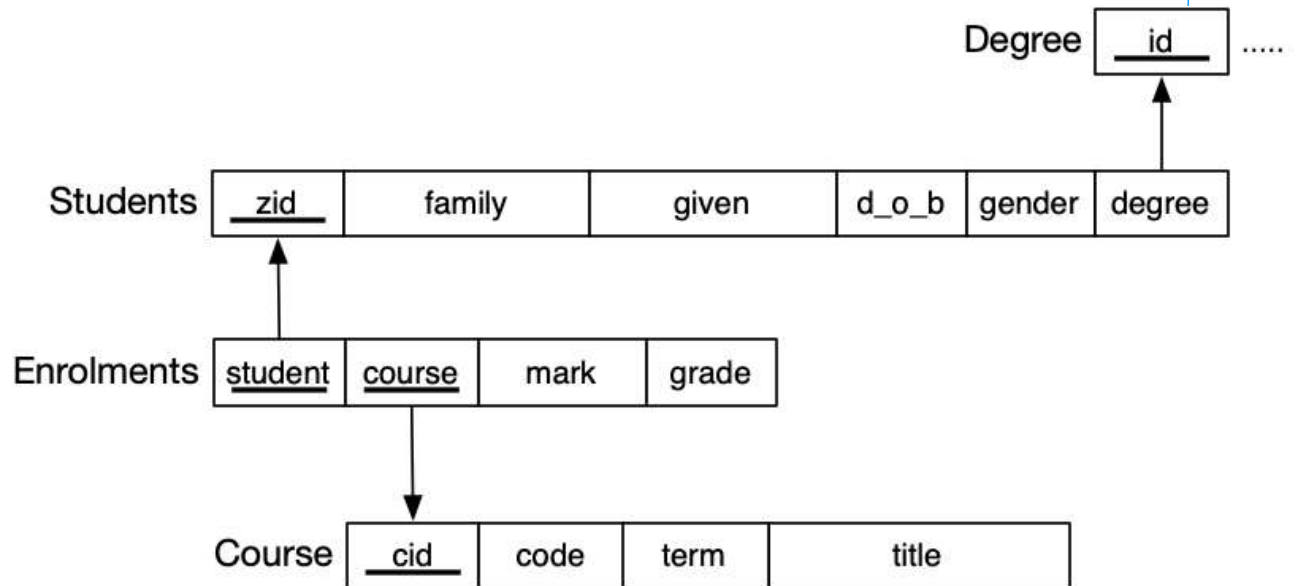
A database schema is a collection of relation schemas that

- defines the structure the whole database
- additional constraints on the whole database

schemas are like your blueprints

## ❖ Example Relational Schema

So far, we have given relational schemas informally, e.g.



COMP3311 20T3 ♦ SQL-DDL ♦ [2/17]

## ❖ SQL Data Definition Language

In the example schema above, we provided only

- relation names, attribute names, primary keys, foreign keys

A usable database needs to provide much more detail

SQL has a rich data definition language (DDL) that can describe

- names of tables
- names and domains for attributes
- various types of constraints (e.g. primary/foreign keys)

It also provides mechanisms for performance tuning (see later).

## ❖ Defining a Database Schema

Tables (relations) are described using:

```
CREATE TABLE TableName (  
    attribute1    domain1    constraints1,  
    attribute2    domain2    constraints2,  
    ...  
    table-level constraints, ...  
)
```

This SQL statement ..

- defines the table schema (adds it to database meta-data)
- creates an empty instance of the table (zero tuples)

Tables are removed via **DROP TABLE *TableName*;**

## ❖ Defining a Database Schema (cont)

**Example:** defining the **Students** table ...

```
CREATE TABLE Students (  
    zid      serial,  
    family   varchar(40),  
    given    varchar(40) NOT NULL,  
    d_o_b    date NOT NULL,  
    gender   char(1) CHECK (gender in ('M','F')),  
    degree   integer,  
    PRIMARY KEY (zid), automatically UNIQUE NOT NULL  
    FOREIGN KEY (degree) REFERENCES Degrees(did)  
);
```

Note that there is much more info here than in the relational schema diagram.

A primary key attribute is implicitly defined to be **UNIQUE** and **NOT NULL**

## ❖ Defining a Database Schema (cont)

Example: alternative definition of the **Students** table ...

```
CREATE DOMAIN GenderType AS
    char(1) CHECK (value in ('M', 'F'));

CREATE TABLE Students (
    zid      serial PRIMARY KEY,
            -- only works if primary key is one attr
    family   text,    -- no need to worry about max length
    given    text NOT NULL,
    d_o_b    date NOT NULL,
    gender   GenderType,
    degree   integer REFERENCES Degrees(did)
);
```

At this stage, prefer to use the long-form declaration of primary and foreign keys

## ❖ Defining a Database Schema (cont)

Example: defining the **Courses** table ...

```
CREATE TABLE Courses (  
    cid      serial,  
    code     char(8) NOT NULL uhs  
            CHECK (code ~ '[A-Z]{4}[0-9]{4}'),  
    term     char(4) NOT NULL  
            CHECK (term ~ '[0-9]{2}T[0-3]'),  
    title    text UNIQUE NOT NULL,  
    PRIMARY KEY (cid)  
);
```

Uses non-standard regular expression checking on **code** and **term**

No two **Courses** can have the same title; but not used as primary key



## ❖ Defining a Database Schema (cont)

Example: defining the **Enrolments** relationship ...

even though we are modelling a relationship, we still use a TABLE

```
CREATE TABLE Enrolments (  
    student integer,  
    course integer,  
    mark integer CHECK (mark BETWEEN 0 AND 100),  
    grade GradeType,  
    PRIMARY KEY (student, course),  
    FOREIGN KEY (student)  
        REFERENCES Students(zid)  
    FOREIGN KEY (course)  
        REFERENCES Courses(cid)  
);
```

Could not enforce total participation constraint if e.g. all courses must have > 0 students

It's possible to make a course that has 0 students, in fact, when you make a course it has 0 by default because there will be no enrolments tuples with that course in it yet.

Possible alternative names for foreign keys **student\_id** and **course\_id**

COMP3311 20T3 ♦ SQL-DDL ♦ [8/17]

Basically, when you use many-to-many, you cannot enforce total participation because the relationship table is separate from the entity itself, and so creating one doesn't force creating the other.

## ❖ Data Integrity

Defining tables as above affects behaviour of DBMS when changing data

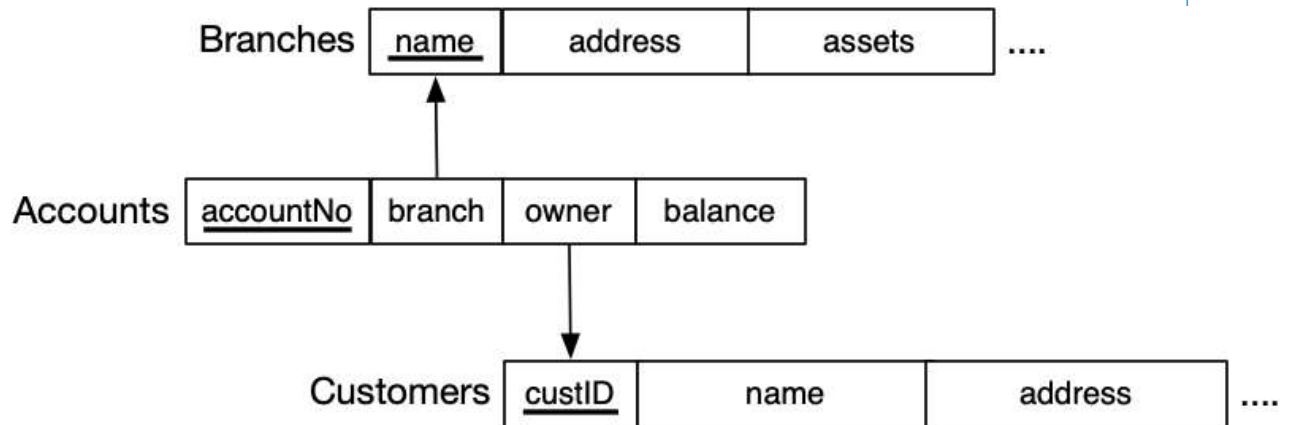
Constraints and types ensure that integrity of data is preserved

- **no duplicate keys** PRIMARY KEY handles this, ensures this PK value does not already exist in the table before insertion is allowed.
- **no "dangling references"** FOREIGN KEY handles this, ensures FK is a PK in referenced table before insertion is allowed. It checks for the actual instance e.g. if FK is 3, then it will look for a PK of 3 in the referenced table b4 insert
- **all attributes have valid values** NOT NULL, UNIQUE, integer, varchar etc.
- etc. etc. etc.

Preserving data integrity is a *critical* function of a DBMS.

## ❖ Another Example Schema

Variation on banking schema used elsewhere



COMP3311 20T3 ♦ SQL-DDL ♦ [10/17]

## ❖ Default Values

Can specify a **DEFAULT** value for an attribute

- will be assigned to attribute if no value is supplied during insert

**Example:**

```
CREATE TABLE Accounts (  
    acctNo char(5) PRIMARY KEY,  
    branch varchar(30) REFERENCES Branches(name)  
        DEFAULT 'Central',  
    owner integer REFERENCES Customers(custID),  
    balance float DEFAULT 0.0  
);
```

```
INSERT INTO Accounts(acctNo,owner) VALUES ('A-456',645342)  
-- produces the tuple  
Accounts('A-456', 'Central', 645342, 0.0)
```

## ❖ Defining Keys

### Primary keys:

- if PK is one attribute, can define as attribute constraint
- if PK is multiple attributes, must define in table constraints
- PK implies **NOT NULL UNIQUE** for all attributes in key

### Foreign keys:

- if FK is one attribute, can define as attribute constraint
- can omit **FOREIGN KEY** keywords in attribute constraint
- if FK has multiple attributes, must define as a single table constraint
- should always specify corresponding PK attribute in FK constraint, e.g

customer integer

FOREIGN KEY REFERENCES Customers(customerNo)



## ❖ Defining Keys (cont)

Defining primary keys assures entity integrity

- must give values for all attributes in the primary key

For example this insertion would fail ..

```
INSERT INTO Enrolments(student, course, mark, grade)
VALUES (5123456, NULL, NULL, NULL);
```

because no **course** was specified; but **mark** and **grade** can be **NULL**

Defining primary keys assures uniqueness

- cannot insert a tuple which contains an existing PK value

## ❖ Defining Keys (cont)

Defining foreign keys assures referential integrity.

On insertion, cannot add a tuple where FK value does not exist as a PK

For example, this insert would fail...

```
INSERT INTO Accounts(acctNo, owner, branch, balance)
VALUES ('A-123', 765432, 'Nowhere', 5000);
```

if there is no customer with id 765432 or no branch Nowhere

COMP3311 20T3 ♦ SQL-DDL ♦ [14/17]

## ❖ Defining Keys (cont)

On deletion, interesting issues arise, e.g.

**Accounts . branch** refers to primary key **Branches . name**

If we want to delete a tuple from **Branches**, and there are tuples in **Accounts** that refer to it, we could ...

- **reject** the deletion (PostgreSQL/Oracle default behaviour)
- **set-NULL** the foreign key attributes in **Account** records
- **cascade** the deletion and **remove Account** records

SQL allows us to choose a strategy appropriate for the application



## ❖ Attribute Value Constraints

**NOT NULL** and **UNIQUE** are special constraints on attributes.

SQL has a general mechanism for specifying attribute constraints

```
attrName type CHECK ( Condition )
```

*Condition* is a boolean expression and can involve other attributes, relations and **SELECT** queries.

```
CREATE TABLE Example
(
    gender char(1) CHECK (gender IN ('M', 'F')),
    Xvalue integer NOT NULL,
    Yvalue integer CHECK (Yvalue > Xvalue),
    Zvalue float CHECK (Zvalue >
                        (SELECT MAX(price)
                         FROM Sells)
    )
);
```

(but many RDBMSs (e.g. Oracle and PostgreSQL) don't allow **SELECT** in **CHECK**)

## ❖ Named Constraints

A constraint in an SQL table definition can (optionally) be named via

```
CONSTRAINT constraintName constraint
```

Example:

```
CREATE TABLE Example
(
    gender char(1) CONSTRAINT GenderCheck
                        CHECK (gender IN ('M','F')),
    Xvalue integer NOT NULL,
    Yvalue integer CONSTRAINT XYOrder
                        CHECK (Yvalue > Xvalue),
);
```

Produced: 21 Sep 2020