Schedule

• Today: Jan. 29 (T)
  ◆ Modifications, Schemas, Views.
  ◆ Read Sections 6.5-6.7. Assignment 3 due.

• Jan. 31 (TH)
  ◆ Constraints.
  ◆ Read Sections 7.1-7.3, 7.4.1. Project Part 3 due.

• Feb. 5 (T)
  ◆ Triggers, PL/SQL.
  ◆ Read Sections 7.4, 8.2. Assignment 4 due.

• Feb. 7 (TH)
  ◆ PL/SQL, Embedded SQL, CLI, JDBC.
  ◆ Read Sections 8.1, 8.3-8.5.

• Reminder: Midterm is Feb. 14 (TH)
Defining a Database Schema

CREATE TABLE name (list of elements).

• Principal elements are attributes and their types, but key declarations and constraints also appear.

• Similar CREATE X commands for other schema elements X: views, indexes, assertions, triggers.

• “DROP X name” deletes the created element of kind X with that name.

Example

CREATE TABLE Sells (  
    bar CHAR(20),  
    beer VARCHAR(20),  
    price REAL  
) ;  

DROP TABLE Sells;
Types

1. INT or INTEGER.
2. REAL or FLOAT.
3. CHAR($n$) = fixed length character string, padded with “pad characters.”
4. VARCHAR($n$) = variable-length strings up to $n$ characters.
   - Oracle uses VARCHAR2($n$) as well.
   - PostgreSQL uses VARCHAR and does not support VARCHAR2.
5. **NUMERIC** (*precision, decimal*) is a number with *precision* digits with the decimal point *decimal* digits from the right. **NUMERIC** *(10,2)* can store ±99,999,999.99

6. **DATE**. SQL form is **DATE** 'yyyy-mm-dd'
   - PostgreSQL follows the standard. Oracle uses a different format.

7. **TIME**. Form is **TIME** 'hh:mm:ss[.ss...]' in SQL.

8. **DATETIME** or **TIMESTAMP**. Form is **TIMESTAMP** 'yyyy-mm-dd hh:mm:ss[.ss...]’ in SQL.

9. **INTERVAL**. Form is **INTERVAL** 'n period' in PostgreSQL. *Period* is month, days, year, etc.
PostgreSQL Dates

PostgreSQL supports extensive date calculations.

- Conversions `to_date(text), to_char(date/time/etc.), interval(text)`
- Date ± Integer = Date;
  Date – Date = Integer (always = number of days);
  Date + Date is invalid!
- Timestamp ± Interval = Timestamp;
  Timestamp – Timestamp = Interval;
  Interval ± Interval = Interval;
  Date + Date is invalid.
- Interval: '1 month' could be 28, 29, 30, or 31 days;
  '31 days' is always just that.
- SQL uses `DATEADD` and `DATEDIFF`;
  PostgreSQL uses the simpler + and −.
- Also `CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP`. 
Declaring Keys

Use **PRIMARY KEY** or **UNIQUE**.

- But only one primary key, many **UNIQUEs** allowed.
- SQL permits implementations to create an *index* (data structure to speed access given a key value) in response to **PRIMARY KEY** only.
  - But PostgreSQL and Oracle create indexes for both.
- SQL does not allow nulls in primary key, but allows them in “unique” columns (which may have two or more nulls, but not repeated non-null values).
Declaring Keys

Two places to declare:

1. After an attribute’s type, if the attribute is a key by itself.

2. As a separate element.
   - Essential if key is >1 attribute.
Example

CREATE TABLE Sells (  
   bar CHAR(20),  
   beer VARCHAR(20),  
   price REAL,  
   PRIMARY KEY(bar,beer)  
);
Example

CREATE TABLE Sells (  
    bar CHAR(20),  
    beer VARCHAR(20),  
    price REAL,  
    UNIQUE(bar,beer)  
);  

is different than:

CREATE TABLE Sells (  
    bar CHAR(20) UNIQUE,  
    beer VARCHAR(20) UNIQUE,  
    price REAL  
);
Other Properties You Can Give to Attributes

1. **NOT NULL** = every tuple must have a real value for this attribute.

2. **DEFAULT value** = a value to use whenever no other value of this attribute is known.

**Example**

```sql
CREATE TABLE Drinkers (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50)
    DEFAULT '123 Sesame St',
    phone CHAR(16)
);```
INSERT INTO Drinkers(name) 
VALUES('Sally')
results in the following tuple:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>123 Sesame St.</td>
<td>NULL</td>
</tr>
</tbody>
</table>

- Primary key is by default not NULL.
- This insert is legal.
  - OK to list a subset of the attributes and values for only this subset.
- But if we had declared
  
  phone CHAR(16) NOT NULL
  
then the insertion could not be made.
Interesting Defaults

- DEFAULT CURRENT_TIMESTAMP
- SEQUENCE

```sql
CREATE SEQUENCE customer_seq;
CREATE TABLE Customer (  
customerID INTEGER DEFAULT nextval('customer_seq'),  
name VARCHAR(30)  
);
```
Changing Columns

Add an attribute of relation $R$ with

```sql
ALTER TABLE $R$ ADD <column declaration>;
```

Example

```sql
ALTER TABLE Bars ADD phone CHAR(16)
   DEFAULT 'unlisted';
```

- Columns may also be dropped.

```sql
ALTER TABLE Bars DROP license;
```
Views

An expression that describes a table without creating it.

- View definition form is:

  ```sql
  CREATE VIEW <name> AS <query>;
  ```
Example

The view `CanDrink` is the set of drinker-beer pairs such that the drinker frequents at least one bar that serves the beer.

```sql
CREATE VIEW CanDrink AS
    SELECT drinker, beer
    FROM Frequents, Sells
    WHERE Frequents.bar = Sells.bar;
```

Querying Views

Treat the view as if it were a materialized relation.

Example

```sql
SELECT beer
FROM CanDrink
WHERE drinker = 'Sally';
```
Semantics of View Use

SQL query $\rightarrow$ rel. algebra $\rightarrow$ SQL

SQL view def. $\rightarrow$ rel. algebra

Example

$\pi_{drinker, beer}\left(\bowtie^{\sigma_{drinker='Sally'}懋频ents \times Sells}\right)$

$\pi_{beer}\left(\bowtie^{\sigma_{drinker='Sally'}懋频ents \times Sells}\right)$

CanDrink

CanDrink

Query
Compose

\[
\pi_{\text{beer}} \\
\sigma_{\text{drinker} = 'Sally'} \\
\pi_{\text{drinker, beer}} \\
\bowtie
\]

\[
\text{Frequents} \quad \text{Sells}
\]
Optimize Query

1. Push selections down tree.
2. Eliminate unnecessary projections.