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;
```
Commercial relational systems allow much more “fine-tuning” of constraints than do the modeling languages we learned earlier.

• In essence: SQL programming is used to describe constraints.

Outline

1. Primary key declarations.
2. Foreign-keys = referential integrity constraints.
3. Attribute- and tuple-based checks = constraints within relations.
4. SQL Assertions = global constraints.
   – Not found in Oracle.
5. Oracle Triggers.
   – A substitute for assertions.

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

```sql
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

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

CREATE SEQUENCE customer_seq;
CREATE TABLE Customer (  
customerID INTEGER
    DEFAULT
nextval('customer_seq'),
    name VARCHAR(30)
);
Foreign Keys

In relation $R$ a clause that “attribute $A$ references $S(B)$” says that whatever values appear in the $A$ column of $R$ must also appear in the $B$ column of relation $S$.
- $B$ must be declared the primary key for $S$.

**Example**

```sql
CREATE TABLE Beers (  
    name CHAR(20) PRIMARY KEY,  
    manf CHAR(20)  
);  

CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20) REFERENCES Beers(name),  
    price REAL  
);  
```

Alternative: add another element declaring the foreign key, as:

```sql
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    FOREIGN KEY beer REFERENCES Beers(name)  
);  
```
- Extra element essential if the foreign key is more than one attribute.
What Happens When a Foreign Key Constraint is Violated?

Two ways:
1. Insert or update a Sells tuple so it refers to a nonexistent beer.
   - Always rejected.
2. Delete or update a Beers tuple that has a beer value some Sells tuples refer to.
   a) Default: reject.
   b) Cascade: Ripple changes to referring Sells tuple.

Example

- Delete “Bud.” Cascade deletes all Sells tuples that mention Bud.
- Update “Bud” to “Budweiser.” Change all Sells tuples with “Bud” in beer column to be “Budweiser.”

- Set Null: Change referring tuples to have NULL in referring components.

Example

- Delete “Bud.” Set-null makes all Sells tuples with “Bud” in the beer component have NULL there.
- Update “Bud” to “Budweiser.” Same change.
Selecting a Policy

Add ON [DELETE, UPDATE] [CASCADE, SET NULL] to declaration of foreign key.

Example

```sql
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    FOREIGN KEY beer REFERENCES Beers(name)  
      ON DELETE SET NULL  
      ON UPDATE CASCADE  
);  
```

- “Correct” policy is a design decision.
  - *E.g.*, what does it mean if a beer goes away? What if a beer changes its name?

Attribute-Based Checks

Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.

- **Form**: CHECK (condition).
  - Condition may involve the checked attribute.
  - Other attributes and relations may be involved, but *only* in subqueries.
  - Oracle: *No subqueries allowed in condition.*
- Condition is checked only when the associated attribute changes (*i.e.*, an insert or update occurs).
Example

CREATE TABLE Sells ( 
    bar CHAR(20),
    beer CHAR(20) CHECK(
        beer IN (SELECT name
               FROM Beers)
    ),
    price REAL CHECK(
        price <= 5.00
    )
);

- Check on `beer` is like a foreign-key constraint, except:
  - The check occurs only when we add a tuple or change the beer in an existing tuple, not when we delete a tuple from `Beers`.

Tuple-Based Checks

Separate element of table declaration.
- Form: like attribute-based check.
- But condition can refer to any attribute of the relation.
  - Or to other relations/attributes in subqueries.
  - Again: Oracle forbids the use of subqueries.
- Checked whenever a tuple is inserted or updated.
Example

Only Joe's Bar can sell beer for more than $5.

CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    CHECK(bar = 'Joe''s Bar' OR price <= 5.00) 
);
Example

No bar may charge an average of more than $5 for beer.

Sells(bar, beer, price)

CREATE ASSERTION NoRipoffBars
CHECK(NOT EXISTS(
    SELECT bar
    FROM Sells
    GROUP BY bar
    HAVING 5.0 < AVG(price)
    )
);

• Checked whenever Sells changes.

Example

There cannot be more bars than drinkers.

Bars(name, addr, license)
Drinkers(name, addr, phone)

CREATE ASSERTION FewBar
CHECK(
    (SELECT COUNT(*) FROM Bars) <=
    (SELECT COUNT(*) FROM Drinkers)
);

• Checked whenever Bars or Drinkers changes.
Types

1. INT or INTEGER.
2. REAL or FLOAT.
3. CHAR\( (n) \) = fixed length character string, padded with “pad characters.”
4. CHARACTER VARYING (or VARCHAR) \( (n) \) = variable-length strings up to \( n \) characters.
5. NUMERIC\( (\text{precision}, \text{decimal}) \) is a number with \text{precision} digits with the decimal point \text{decimal} digits from the right. \text{NUMERIC}(10, 2)\ can store \pm 99,999,999.99
Types

6. **DATE.** SQL form is DATE 'yyyy-mm-dd'
   - Oracle uses a different standard format
   - Use “Alter session” command to fix it
7. **TIME.** Form is TIME 'hh:mm:ss[.ss...]' in SQL.
8. **DATETIME or TIMESTAMP.** Form is TIMESTMPP 'yyyy-mm-dd hh:mm:ss[.ss...]' in SQL.
   - Generally preferable to just “Date” or “time”
   - Be careful with semantics of comparison
9. **Various extensions**
   - INTERVAL – time interval
   - BLOB, LONG, RAW – large objects with little typing

Changing Columns

Add an attribute of relation \( R \) with

\[
\text{ALTER TABLE } R \text{ ADD } <\text{column declaration}>; 
\]

**Example**

\[
\text{ALTER TABLE Bars ADD phone CHAR(16) DEFAULT 'unlisted'}; 
\]

- Columns may also be dropped.

\[
\text{ALTER TABLE Bars DROP license; } 
\]
Views

An expression that describes a table without creating it.

- View definition form is:
  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';
```
Theory behind views

• Every relational query returns a relation
  – Possibly a single row, single column relation
• Query result could be stored in a table
  – Use in future queries
• View: Do this “on the fly”
  – Generate the result *every time the view is used*

Using Views

• Access control: Limit who sees data
• Different logical views
  – Schema migration
• “short cuts”
View Limitations

• Performance
  – Materialized views

• Update
  – Insert
  – Modify
  – Delete

• Solutions to come
  – Triggers