SQL

Data Definition Language

Orlando Karam
okaram@spsu.edu

Data Definition Language

- Subset of SQL
- Create and manage Tables, Views, Indexes and other Database Objects
  - CREATE
  - DROP
  - ALTER
- Kinds of Objects
  - TABLE
  - VIEW
  - INDEX
  - SEQUENCE, ...

CREATE TABLE

```
CREATE TABLE table_name (
 field type constraints,
 field2 type2 ,
 CONSTRAINT name ...,
 ...
);
```

```
CREATE TABLE Book (
   ISBN  CHAR(9)      PRIMARY KEY,
   Title VARCHAR(20) UNIQUE NOT NULL,
   Pages Integer
);
```

Common Datatypes

- CHAR(n)
  - fixed length strings, padded with spaces at end
- VARCHAR(n)
  - variable length strings, but no longer than n
  - Oracle mentions VARCHAR2 ...
- NUMERIC(prec,dec) -- oracle NUMBER(p,d)
  - fixed precision numbers (not floats)
  - precision is **total** number of digits, dec is how many after the decimal point
  - Auto-increment done with sequences
- DATE, TIMESTAMP
  - Represent dates, or specific points in time

Common constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- REFERENCES (foreign key)
  - after REFERENCES put name of table, then field in parenthesis
  - StudentId REFERENCES Student(Id)
- CHECK
  - Allows for predicates after
  - CHECK(age>20)

CREATE TABLE Example

```
CREATE TABLE Student (
  Id CHAR(3) PRIMARY KEY,
  Name VARCHAR(20) NOT NULL,
  Age CHECK(Age>0 AND AGE<100),
  Gender CHAR NOT NULL,
  Deg_code CHAR(2) NOT NULL REFERENCES Degree(code),
  Major CHAR(3),
  credits INTEGER
)
;
```

More on Constraints

- Can go at end of table, for naming and for multi-column constraints
  - –CONSTRAINT my_pk PRIMARY KEY(user,email)

```
CREATE TABLE Standing (
  deg_code       char(2) REFERENCES Degree(deg_code),
  min_cr         INTEGER DEFAULT 0 NOT NULL,
  max_cr         INTEGER NOT NULL,
  num            INTEGER NOT NULL,
  designation    VARCHAR(20) NOT NULL,
  CONSTRAINT Standing_PK
        PRIMARY KEY (deg_code, num),
  CONSTRAINT Standing_Unique_Designation
        UNIQUE (deg_code, designation),
  CONSTRAINT Standing_min_max
        CHECK (min_cr <= max_cr)
);
```

Virtual Columns (11g specific)

- Can define a calculated column (instead of using views or triggers)

REFERENCE (foreign keys)

- CONSTRAINT my_fk FOREIGN KEY (fk_field) REFERENCES other_table(id)
- ON DELETE (Oracle doesn't support ON UPDATE
  - RESTRICT
  - CASCADE
  - SET NULL
  - SET DEFAULT
- Deferring
  - SET CONSTRAINT ALL DEFERRED/IMMEDIATE

Example

- Imagine we have people, with a table for (multivalued) emails.

```
CREATE TABLE Person (
  id integer primary key,
  name varchar(20)
);
```

```
CREATE TABLE Emails (
  person_id integer CONSTRAINT e_fk REFERENCES Person(id) ,
  email varchar(20) NOT NULL CHECK (email LIKE '%@%.%'),
  CONSTRAINT Emails_PK PRIMARY KEY (Person_id,Email)
);
```

```
CREATE TABLE Emails1 (
  person_id integer CONSTRAINT e1_fk REFERENCES Person(id)
                    ON DELETE CASCADE,
  email varchar(20) NOT NULL CHECK (email LIKE '%@%.%'),
  CONSTRAINT Emails1_PK PRIMARY KEY (Person_id,Email)
);
```

DROP TABLE

- Deletes the table, and all data it contained.
- CASCADE CONSTRAINTS
- PURGE

ALTER TABLE

- Changes columns/constraints on a particular table
- Data automatically ‘translated’ if possible.
- ALTER TABLE name
  - –ADD name VARCHAR(2)
  - –ADD CONSTRAINT ttt PRIMARY KEY (name)

VIEWS

- Act as named queries
- Can use basically all the features of a SELECT statement
  - But we'll be defining simple ones now :)
- Updating data in the view is problematic
  - what does it mean ? For complicated ones ?
  - But we can define with triggers
- Calculated vs Materialized views
- ALTER VIEW xxx COMPILE

View example

- CREATE VIEW women
- AS
- SELECT *
- FROM Person
- WHERE Gender=’F’

- After, I can do like:
  - –SELECT *
  - –FROM Women

Indexes

- Help make searches faster
  - –but have space overhead
  - –and may make changes to tables slower
  - –only accelerate searches on indexed field
- CREATE INDEX my_index ON Person(id)
- Automatically created for primary key fields

SEQUENCES

- Transaction-safe auto-increments
  - but can have ‘holes’
- CREATE SEQUENCE xyz;
- SELECT xyz.nextval FROM dual;

SEQUENCE Example

- CREATE SEQUENCE PersonSequence
- INCREMENT BY 1
- START WITH 50;

- INSERT INTO Person(id,name) VALUES (PersonSequence.nextval,'John');

- SELECT * from person;

- SELECT PersonSequence.nextval
- FROM Dual;