CMSC 508
Database Theory

Advanced SQL (III)

Dr. Alberto Cano
Assistant Professor
Department of Computer Science

Chapter 4 from Database System Concepts, 6th Ed. by Silberschatz, Korth, Sudarshan, 2011
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003

- Sequences
  - Database objects from which users may generate unique integers. Use sequences to automatically generate primary key values.
  - The sequence of numbers can be generated in either ascending or descending order.
  - Optional arguments: MAXVALUE, MINVALUE, CYCLE, CACHE, ORDER ...

```
CREATE SEQUENCE name_of_my_seq
 START WITH     1000
 MAXVALUE       5000
 INCREMENT BY   1
 NOCYCLE;
```

  - Call the next value anytime as:

```
name_of_my_seq.nextval
```

- Functions and procedures
  - Users can define new methods (functions and procedures) to be called in a SQL statement, trigger, etc.
  - Functions are typically coded to perform a small calculation.
  - Procedures are typically coded to perform larger operations implying database changes.
  - **Difference**: procedures can be called with SQL statements, while functions are called as **part of** an expression. Functions return values to the caller environment.

- User-defined functions
  - Can accept input parameters (optional) and return a data type
  - In functions, a RETURN statement **must** contain an expression
  - Functions **CANNOT modify** data in any table

```
CREATE [OR REPLACE] FUNCTION <function_name> (
            <parameter1_name> <data type>,
            <parameter2_name> <data type>, ...)
RETURN <function return value data type> {AS|IS}
   <Variable declarations>
BEGIN
   Executable Commands
   RETURN (return_value);
   . . .
[EXCEPTION
   Exception handlers]
END;
```

- User-defined functions
- Functions with no parameters

```
CREATE OR REPLACE FUNCTION RetrieveSalary
    RETURN NUMBER
IS
    v_Salary NUMBER(10,2);
BEGIN
    SELECT salary INTO v_Salary
    FROM employees
    WHERE employee_id = '100';
    RETURN v_Salary;
END RetrieveSalary;
/
```

- Functions with parameters

```
CREATE OR REPLACE FUNCTION RetrieveSalary
( p_employee_id in
  employees.employee_id%TYPE )
    RETURN NUMBER
IS
    v_Salary employees.salary%TYPE;
BEGIN
    SELECT salary INTO v_Salary
    FROM employees
    WHERE employee_id = p_employee_id;
    RETURN v_Salary;
END RetrieveSalary;
/
```

- Procedures
  - In procedures, a RETURN statement **cannot** contain an expression.

```
CREATE [OR REPLACE] PROCEDURE <procedure_name> (
<parameter1_name> <data type>,
<parameter2_name> <mode> <data type>, ...)
{AS|IS}
   <Variable declarations>
BEGIN
   Executable statements
[EXCEPTION
   Exception handlers]
END <optional procedure name>;
```

```
CREATE OR REPLACE PROCEDURE
IncreaseSalary (
  p_employee_id in
employees.employee_id%TYPE )
IS
BEGIN
    UPDATE employees set salary = salary * 1.05
    where employee_id = p_employee_id;
END IncreaseSalary;
/
```

- Cursors (iterators)

```
create or replace PROCEDURE get_emp_names (dept_num IN NUMBER)
IS
    emp_name  VARCHAR2(10);
    CURSOR    c1 (dept_num NUMBER) IS
              SELECT LAST_NAME FROM EMPLOYEES
              WHERE DEPARTMENT_ID = dept_num;
BEGIN
  OPEN c1(dept_num);
  LOOP
    FETCH c1 INTO emp_name;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_name);
  END LOOP;
  CLOSE c1;
END;
/
```

- Advanced table creation: Integrity constraints
  - Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency
  - Constraints in a single relation:
    - **primary key**
    - **not null**
    - **unique**
    - **check** (P), where P is a predicate

- Integrity constraints
  - **not null**

```
create table students (
        last_name varchar2(20) not null
);
```

  - **unique**

```
create table students (
        last_name varchar2(20) not null,
        email varchar2(20) not null unique
);
```

- Integrity constraints
  - **primary key**

```
create table students (
        id number(6,0) primary key,
        last_name varchar2(20) not null,
        email varchar2(20) not null unique
);

create table students (
        first_name varchar2(20) not null,
        last_name varchar2(20) not null,
        email varchar2(20) not null unique,
        primary key (first_name, last_name)
);
```

- Integrity constraints
  - **check** (P), where P is a predicate

```
create table students (
        first_name varchar2(20) not null,
        last_name varchar2(20) not null,
        email varchar2(20) not null unique,
        semester varchar2(20),
        primary key (first_name, last_name),
        check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

However, we **cannot** include a subquery in a check predicate:

```
check (email not in (select email from spam_addresses))
```

Therefore, use triggers to control such “dynamic” behavior!

- Referential integrity
  - Foreign keys

```
create table employee (
        department_id number(4,0)
                references departments
                on update cascade on delete cascade
);
```

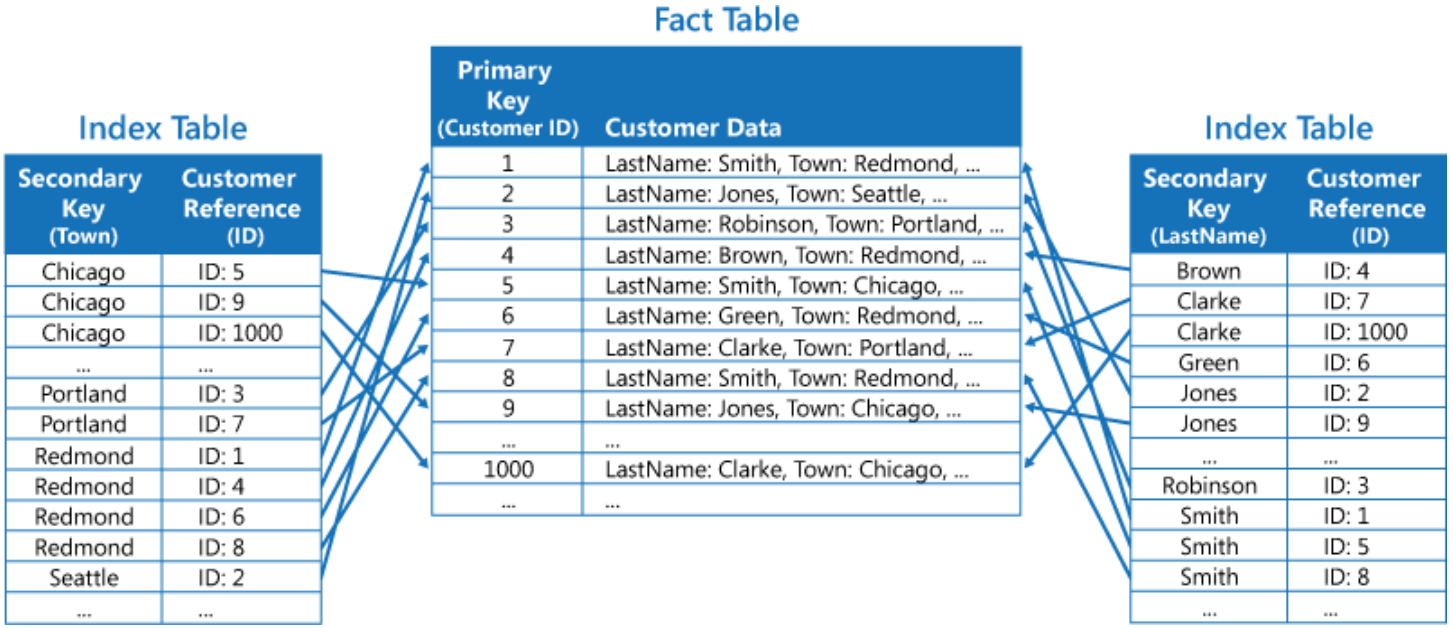
  - **CASCADE**: Delete or update the row from the parent table, and **automatically** delete or update the matching rows in the child table
  - **Referential actions**: cascade, set null, set default, no action

- Data types
  - **create type** construct in SQL creates user-defined type

```
CREATE OR REPLACE TYPE car AS OBJECT
(
        price number(8,0),
        model varchar2(20)
);
```

  - DB designer decision, data representation: relation vs object.
  - Most useful in object-oriented DBMS.

- Indices
  - An **index** is a schema object that contains an entry for each value that appears in the indexed column(s) of the and provides direct, fast access to rows



Syntax: **create index** *myindexfooname* **on** *table*(column);
Example: **create index** *jobID_index* **on** *employees*(job_id);

- Exercises

The following exercises, together with the ones from triggers are expected to be uploaded into blackboard:

- Create a sequence to control the PK generation of the employees.
- Create a function to return the full name for an employee whose id is provided as parameter.
- Create a procedure to increase 10% the salary of the manager whose subordinate id is provided as parameter.
- Create a table for projects (manager, duration (days), cost), and check that the cost must be < 500 per day nor bigger than the sum of the salaries of the department employees the manager works for
- Create a mechanism to check and prevent employees salary bigger than his manager (or King if they have no manager).

CMSC 508
Database Theory

Advanced SQL (III)

Dr. Alberto Cano
Assistant Professor
Department of Computer Science

Chapter 4 from Database System Concepts, 6th Ed. by Silberschatz, Korth, Sudarshan, 2011
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003