Lecture 2 Notes

Relational Model

Discussion Points:

- Data Representation
- Integrity Constraints
- SQL: Creating and Managing Tables
- SQL: Constraints

❖ Data Representation

The main construct for representing data in the relational model is a relation.

- *Relation:* A relation consists of a relation schema and a relation instance.
- *Relation Schema:* Relation schema describes the column heads for the table.
  Eg:
  Students (sid: varchar, name: varchar, login: varchar, age: integer, gpa: float)
- *Relation Instance:* Relation instance is a table.
- *Relational Database:* A relational database is a collection of relations with distinct relation names.
  Eg:

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	glldll@music	12	2.0
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53666	Jones	jones@cs	18	3.4
50000	Dave	dave@cs	19	3.3

- *Degree:* The degree, also called arity, of a relation is the number of fields.
- *Cardinality:* The cardinality of a relation instance is the number of tuples in it.
- In above Figure, the degree of the relation (the number of columns) is five, and the cardinality of this instance is six.

❖ Integrity Constraints

- An integrity constraint is a condition specified on a database schema and restricts the data that can be stored in an instance of the database.

- *Legal Instance:* If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

- *Super Key:* A superkey of a relation is a set of one or more attributes whose values are guaranteed to identify tuples in the relation uniquely. A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
  For example, the ID attribute of the relation instructor is sufficient to distinguish one instructor tuple from another. Thus, ID is a superkey. The name attribute of instructor, on the other hand, is not a superkey, because several instructors might have the same name.

- *Candidate Key:* A candidate key is a minimal superkey, that is, a set of attributes that forms a superkey, but none of whose subsets is a superkey. A superkey may contain extraneous attributes. Candidate key must satisfy these two constraints:

  1. Two distinct tuples in a legal instance cannot have identical values in all the fields of a key.

  2. No subset of the set of fields in a key is a unique identifier for a tuple.

For example, the combination of ID and name is a superkey for the relation instructor. If K is a superkey, then so is any superset of K. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys. It is possible that several distinct sets of attributes could serve as a candidate key.

- *Primary Key:* Primary key is used to denote a candidate key that is chosen by the database designer as the principal means of uniquely identifying tuples within a relation. One of the candidate keys of a relation is chosen as its primary key.

- *Alternate Key:* An alternate is a secondary candidate key that is capable of identifying a row uniquely.

- *Foreign Key:* A relation, say r1, may include among its attributes the primary key of another relation, say r2. This attribute is called a foreign key from r1, referencing r2. The relation r1 is also called the *referencing relation* of the foreign key dependency, and r2 is called the *referenced relation* of the foreign key.

- *Referential Integrity Constraint:* A referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

❖ SQL: Creating and Managing Tables

- *Database Objects*
  - *Table:* Basic unit of storage; composed of rows and columns.
  - *View:* Logically represents subsets of data from one or more tables.
  - *Sequence:* Numeric value generator.
  - *Index:* Improves the performance of some queries.
  - *Synonym:* Gives alternative names to objects.

- *DDL Statements*
  - CREATE TABLE: Creates a table
  - ALTER TABLE: Modifies table structures
  - DROP TABLE: Removes the rows and table structure
  - RENAME: Changes the name of a table, view, sequence, or synonym
  - TRUNCATE: Removes all rows from a table and releases the storage space
  - COMMENT: Adds comments to a table or view

- *Naming Rules (Tables and Columns)*
  - Must begin with a letter
  - Must be 1–30 characters long
  - Must contain only A–Z, a–z, 0–9, _, $, and #
  - Must not duplicate the name of another object owned by the same user
  - Must not be a reserved word
- *User Tables*
  - Are a collection of tables created and maintained by the user
  - Contain user information

- *Data Dictionary*
  - Is a collection of tables created and maintained by the Server
  - Contain database information

- *Datatypes*

Data Type	Description
VARCHAR2(*size*)	Variable-length character data
CHAR(*size*)	Fixed-length character data
NUMBER(*p*,*s*)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data up to 2 gigabytes
CLOB	Character data up to 4 gigabytes
RAW and LONG RAW	Raw binary data
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes
ROWID	A 64 base number system representing the unique address of a row in its table.

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days to hours minutes and seconds

- *CREATE Table*

```
CREATE TABLE [schema.] table
        (column datatype [DEFAULT expr] [, ...]);
```

Eg:

```
CREATE TABLE dept
        (deptno NUMBER(2),
        dname VARCHAR2(14),
        loc VARCHAR2(13));

DESCRIBE dept;
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

- *Creating a Table by Using a Subquery Syntax*

```
CREATE TABLE table [(column, column...)]
        AS subquery;
```

Eg:

```
CREATE TABLE dept
    AS
    SELECT employee_id, last_name, salary*12 ANNSAL, hire_date
    FROM employees
    WHERE dept_id = 10;
```

- *ALTER Table*

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP          (column);
```

Eg (ADD New Column):

```
ALTER TABLE dept
ADD     (job_id VARCHAR2(9));
```

Eg (Modify Column data type/ size/add default value.):

```
ALTER TABLE dept
MODIFY          (last_name VARCHAR2(30));
```

Eg (DROP Column):

```
ALTER TABLE dept
DROP COLUMN  job_id;
```

- *DROP Table*
  - *All data and structure in the table is deleted.*
  - *Any pending transactions are committed.*
  - *All indexes are dropped.*
  - *You cannot roll back the DROP TABLE statement.*

```
DROP TABLE table_name;
```

Eg:

```
DROP TABLE dept;
```

- *RENAMING an Object*
  - To change the name of a table, view, sequence, or synonym, you execute the RENAME statement.

```
RENAME {old_name} TO {new_name};
```

Eg:

```
RENAME dept TO detail_dept;
```

- *TRUNCATE TABLE*
  - Removes all rows from a table
  - Releases the storage space used by that table
  - You cannot roll back row removal when using TRUNCATE.
  - Alternatively, you can remove rows by using the *DELETE statement.*

TRUNCATE TABLE table_name;

Eg:
TRUNCATE TABLE detail_dept;

❖ **SQL Constraints**
- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

- The following constraint types are valid:
  - *Column Level*

column [CONSTRAINT constraint_name] constraint_type,

  - *Table Level*

column,...
[CONSTRAINT constraint_name] constraint_type
(column, ...),

- *NOT NULL*
  - NOT NULL constraint: No row can contain a null value for this column.
  - Is defined at the column level.

Eg:
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) **NOT NULL**,
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE
CONSTRAINT emp_hire_date_nn
NOT NULL,
...

- *UNIQUE*
  - UNIQUE constraint: It is a column level constraint used to ensure that all values must be unique within the respective column.
  - It can contain NULL values.
  - Defined at either the table level or the column level.

Eg:
```
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
email VARCHAR2(25),
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE NOT NULL,
...
CONSTRAINT emp_email_uk UNIQUE(email));
```

- *PRIMARY KEY*
  - The PRIMARY KEY constraint uniquely identifies each record in a table.
  - Primary keys must contain UNIQUE values and cannot contain NULL values.
  - A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
  - Defined at either the table level or the column level.

Eg:
```
CREATE TABLE departments(
department_id NUMBER(4),
department_name VARCHAR2(30)
CONSTRAINT dept_name_nn NOT NULL,
manager_id NUMBER(6),
location_id NUMBER(4),
CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

- *FOREIGN KEY*
  - A foreign key is a column (or combination of columns) in a table whose values must match values of a column in some other table.
  - FOREIGN KEY constraints enforce **Referential Integrity,** which essentially says that if column value A refers to column value B, then column value B must exist.

Eg:
```
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
...
department_id NUMBER(4),
CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
REFERENCES departments(department_id),
CONSTRAINT emp_email_uk UNIQUE(email));
```

- *FOREIGN KEY:* Defines the column in the child table at the table constraint level.
- *REFERENCES:* Identifies the table and column in the parent table.
- *ON DELETE CASCADE:* Deletes the dependent rows in the child table when a row in the parent table is deleted.
- *ON DELETE SET NULL:* Converts dependent foreign key values to null.

- *CHECK*
  - Defines a condition that each row must satisfy
  - The following expressions are not allowed:
    - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
    - Calls to SYSDATE, UID, USER, and USERENV functions
    - Queries that refer to other values in other rows

Eg:
..., salary NUMBER(2)
CONSTRAINT emp_salary_min
CHECK (salary > 0),...

- *ASSERTIONS*
  - An assertion is a predicate expressing a condition that we wish the database always to satisfy.
  - Domain constraints and referential-integrity constraints are special forms of assertions.
  - Complex check conditions can be useful when we want to ensure integrity of data, but may be costly to test.
  - An assertion in SQL takes the form:

CREATE ASSERTION <assertion-name> **CHECK** <predicate>;

Eg:
create assertion credits_earned_constraint **check**
(not exists (select ID
from student
where tot cred <> (select sum(credits)
from takes natural join course
where student.ID= takes.ID
and grade is not null and grade<> 'F')

- *Adding a Constraint*
  - Use the ALTER TABLE statement to:
    - Add or drop a constraint, but not modify its structure
    - Enable or disable constraints
    - Add a NOT NULL constraint by using the MODIFY clause

ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);

Eg:

```
ALTER TABLE employees
        ADD CONSTRAINT emp_manager_fk
        FOREIGN KEY(manager_id)
        REFERENCES employees(employee_id);
```

- *Dropping a Constraint*
  - Use the ALTER TABLE statement to:
    - Add or drop a constraint, but not modify its structure
    - Enable or disable constraints
    - Add a NOT NULL constraint by using the MODIFY clause

Eg:

```
ALTER TABLE employees
        DROP CONSTRAINT emp_manager_fk;

ALTER TABLE departments
        DROP PRIMARY KEY CASCADE;
```

- *Viewing a Constraint*
  - Query the USER_CONSTRAINTS table to view all constraint definitions and names.

Eg:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'EMPLOYEES';
```

  - View the columns associated with the constraint names in the USER_CONS_COLUMNS view.

Eg:

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'EMPLOYEES';
```

❖ **References:**

- "Database System Concepts", Avi Silberschatz, Henry F. Korth, S. Sudarshan, McGraw-Hill.
- "Database Management Systems", Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill.
- "Fundamentals of Database Systems", R. Elmasri, S. B. Navathe, Pearson.
- Oracle SQL Resources
- Other Internet Source