Lecture Notes for Lecture 6 of CS 5200 (Database Management System) for the Summer, 2019 session at the Northeastern University Silicon Valley Campus.

Keys and Constraints

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 5 Review

- Entity-relationship data models are a good way to model real-world data as a starting point for a relational representation.
- Various ER design patterns can be transformed into relations by following a formula for the pattern.
- The concept of functional dependencies leads to a set of transformations on relational models normalize the tables in a way that simplify queries
- The three basic transformations from First Normal Form (1NF) to Second Normal Form (2NF) to Third Normal Form (3NF) eliminate common functional dependencies
- In-class demonstrations showed examples of how to perform common database operations using the JDBC APIs using the Wikipedia Employee and Department tables.

Keys and Constraints

Discussion of Assignment 3

- We will begin by discussing different ways to represent the relationships for the ‘publication’ database for assignment 3.
- We will also discuss several of the publication-specific data types and how they are represented, including
  - ISSN (International Standard Serial Number)
  - DOI (Digital Object Identifier)
  - ORCID (ORCHID persistent author identifier)
- Finally we will look a test program and a data file for testing your code.

Keys and Constraints

Testing Database Tables with Data

- Once the database tables are created from the ER model, it is important to test the tables using sample of data extracted from actual data sets.
- This helps ensure that the fields have the correct size and type to accommodate the data they will eventually hold, to identify anomalies in the data that must be pre-processed.
- Create test programs that read the data sets and create the appropriate INSERT statements, rather than creating the INSERT statements by hand.
- The test program can either be written to load the database directly, or to output the INSERT statements for use with a tool like *ij* that executes the INSERT statements.

Keys and Constraints

- An in-class demonstrations shows how to test the tables created from the ER diagrams from Assignment 3 with Publisher, Journal, Article, and Author entities and PublishedBy, PublishedIn, and WrittenBy relations.
- Tests use “live” publication data for articles written for four journals published by two professional societies: the Association for Computing Machinery (ACM) and the Institute for Electrical and Electronics Engineers (IEEE).
- Input data has lines with the nine entity attributes for a combination of publisher, journal, article, and author in a tab-separate format. The test program read the lines and populated the database.

Keys and Constraints

Review of Relational Algebra Constraints

- Every relation has conditions that must hold to be valid.
- Three main types of integrity constraints
  - Key constraints
  - Domain constraints
  - Referential integrity constraints

Keys and Constraints

Key Constraints

- A minimal subset of attributes in a relation can uniquely identify a tuple
- Minimal subset of attributes is known as the *key*.
- If more than one such subset, they are *candidate keys*.
- Key constraints force that:
  - No two tuples have identical values for key attributes
  - Key attribute cannot have NULL value.
- Key constraints also known as *entity constraints*.

Keys and Constraints

Domain Constraints

- Attributes of relation limited by real-world scenarios
- Same attributes that were employed in ER model
- Examples:
  - Age cannot be less than 0
  - Telephone number cannot contain digits outside 0-9
  - Identifier must begin with at least one letter and may have any number of additional letters or digits.

Keys and Constraints

Referential Integrity Constraint

- Referential integrity constraint states that if relation refers to key attribute of another or same relation, then key elements must exist.
- Works on concept of a foreign key: a key attribute of one relation that can be referred to in another relation.

Keys and Constraints

SQL and Constraints

- Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
- Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.
- Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

Keys and Constraints

SQL and Constraints

These are the most commonly used constraints available in SQL.

- NOT NULL Constraint − Ensures that a column cannot have NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX − Used to create and retrieve data from the database very quickly.

Keys and Constraints

NOT NULL Constraint

- By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.
- A NULL is not the same as no data, rather, it represents unknown data.
- Example: the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, ID NAME and AGE, do not accept NULLs

```
CREATE TABLE CUSTOMERS(
        ID INT NOT NULL,
        NAME VARCHAR (20) NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR (25) ,
        SALARY DECIMAL (18, 2)
)
```

Keys and Constraints

NOT NULL Constraint

- If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column by writing a query like this

```
ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) NOT NULL
```

Keys and Constraints

DEFAULT Constraint

- The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.
- Example: the following SQL creates a new table called CUSTOMERS and adds five columns.
- The SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
        ID INT NOT NULL,
        NAME VARCHAR (20) NOT NULL,
        AGE INT NOT NULL, ADDRESS CHAR (25) ,
        SALARY DECIMAL (18, 2) DEFAULT 5000.00,
);
```

Keys and Constraints

DEFAULT Constraint

- If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like these.
- Adding a default value to CUSTOMERS table:

```
MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00
```

- Removing a default value for the CUSTOMERS table:

```
ALTER TABLE CUSTOMERS ALTER COLUMN SALARY DROP DEFAULT
```

Keys and Constraints

UNIQUE Constraint

- The UNIQUE Constraint prevents two records from having identical values in a column. In the CUSTOMERS table, for example, you might want to prevent two or more customers from having an identical addresses.
- Example: the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the ADDRESS column is set to UNIQUE, so that you cannot have two records with the same address.

```
CREATE TABLE CUSTOMERS(
        ID INT NOT NULL,
        NAME VARCHAR (20) NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR (25) UNIQUE ,
        SALARY DECIMAL (18, 2),
)
```

Keys and Constraints

UNIQUE Constraint

- If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the ADDRESS column. You would write a statement like this.

```
ALTER TABLE CUSTOMERS MODIFY ADDRESS CHAR (20) NOT NULL UNIQUE
```

- You can also use the following syntax, which supports naming the constraint in multiple columns as well.

```
ALTER TABLE CUSTOMERS ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

- To drop a UNIQUE constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS DROP CONSTRAINT myUniqueConstraint
```

Keys and Constraints

PRIMARY Key

- A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.
- A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Keys and Constraints

PRIMARY Key

- Here is the syntax to define the ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMERS (
        ID INT NOT NULL,
        NAME VARCHAR (20) NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR (25) ,
        SALARY DECIMAL (18, 2),
        PRIMARY KEY (ID)
)
```

Keys and Constraints

PRIMARY Key

- To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use this SQL syntax.

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

- Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) should have already been declared to not contain NULL values (when the table was first created).

Keys and Constraints

PRIMARY Key

- For defining a PRIMARY KEY constraint on multiple columns, use this SQL syntax.

```
CREATE TABLE CUSTOMERS (
        ID INT NOT NULL,
        NAME VARCHAR (20) NOT NULL,
        AGE INT NOT NULL,
        ADDRESS CHAR (25) ,
        SALARY DECIMAL (18, 2),
        PRIMARY KEY (ID, NAME)
)
```

Keys and Constraints

PRIMARY Key

- To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use this SQL syntax.

```
ALTER TABLE CUSTOMERS ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

- To delete the primary key constraints from the table, use this SQL syntax.

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;

ALTER TABLE CUSTOMERS DROP CONSTRAINT PK_CUSTID;
```

Keys and Constraints

FOREIGN Key

- A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.
- A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
- The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
- If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

Keys and Constraints

FOREIGN Key

- Consider the structure of the following two tables.

```
CREATE TABLE Customer (
        id INT NOT NULL,
        name VARCHAR (20) NOT NULL,
        age INT NOT NULL,
        address CHAR (25) ,
        PRIMARY KEY (id)
)

CREATE TABLE Orders (
        id INT NOT NULL,
        date DATETIME,
        amount double,
        customer_id INT;
        PRIMARY KEY (id)
        FOREIGN KEY(customer_id) references Customers(ID),
)
```

Keys and Constraints

FOREIGN Key

- Note: CUSTOMERS table needs to be created first, followed by the ORDERS table.
- An alternate syntax allows adding a name to a constraint

```
CREATE TABLE Orders (
        id INT NOT NULL,
        date DATETIME,
        amount double,
        customer_id INT;
        PRIMARY KEY (id)
        CONSTRAINT customerid_fk
           FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS (ID)
)
```

Keys and Constraints

FOREIGN Key

- If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE Orders ADD FOREIGN KEY (customer_id) REFERENCES Customers (id);
```

- To drop a FOREIGN KEY constraint, use the following SQL syntax.

```
ALTER TABLE ORDERS DROP FOREIGN KEY;

ALTER TABLE ORDERS DROP FOREIGN KEY(customer_id);
```

Keys and Constraints

CHECK Constraint

- The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.
- Example: the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE Customers (
        id INT NOT NULL,
        name VARCHAR (20) NOT NULL,
        age INT NOT NULL CHECK (age >= 18),
        address CHAR (25) ,
        PRIMARY KEY (id)
)
```

Keys and Constraints

CHECK Constraint

- If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like this.

```
ALTER TABLE Customers MODIFY age INT NOT NULL CHECK (age >= 18 );
```

- You can also use the following syntax, which supports naming the constraint in multiple columns as well.

```
ALTER TABLE Customers ADD CONSTRAINT myCheckConstraint CHECK(age >= 18)
```

- To drop a CHECK constraint, use the following SQL syntax

```
ALTER TABLE Customers DROP CONSTRAINT myCheckConstraint;
```

Keys and Constraints

INDEX

- The INDEX is used to create and retrieve data from the database very quickly.
- An Index can be created by using a single or group of columns in a table. When the index is created, it is assigned a ROWID for each row before it sorts out the data.
- Proper indexes are good for performance in large databases, but you need to be careful while creating an index.
- A selection of fields depends on what you are using in your SQL queries.

Keys and Constraints

INDEX

- For example, the following SQL syntax creates a new table called CUSTOMERS and adds five columns in it.

```
CREATE TABLE Customers (
        id INT NOT NULL,
        name VARCHAR (20) NOT NULL,
        age INT NOT NULL,
        address CHAR (25) ,
        PRIMARY KEY (id)
)
```

- Now, you can create an index on a single or multiple columns using the syntax given below.

```
CREATE INDEX index_name ON table_name ( column1, column2.....)
```

Keys and Constraints

INDEX

- To create an INDEX on the AGE column, to optimize the search on customers for a specific age, you can use the follow SQL syntax.

```
CREATE INDEX idx_age ON Customers ( age );
```

- To drop an INDEX constraint, use the following SQL syntax.

```
ALTER TABLE Customers DROP INDEX idx_age;
```

Keys and Constraints

Auto-Increment Field

- It is possible to add a field that acts as an identity in a row of the database. This is known as an auto-increment field.
- The syntax for an auto-increment field varies by database. Here is how one is specified for Derby for the ID field of the CUSTOMERS table

```
CREATE TABLE CUSTOMERS (
        id INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
        NAME VARCHAR (20) NOT NULL,
        age INT NOT NULL,
        address CHAR (25) ,
        PRIMARY KEY (id)
)
```

- Now data can be inserted as

```
INSERT INTO CUSTOMERS VALUES ( “Joe”, 42, “123 Main St.” )
```

Keys and Constraints

Auto-Increment Field

- Here is the same auto-increment field is specified in MySQL

```
CREATE TABLE Customers (
        id INT NOT NULL AUTO_INCREMENT,
        name VARCHAR (20) NOT NULL,
        age INT NOT NULL,
        address CHAR (25) ,
        PRIMARY KEY (id)
)
```

- To alter Customers to have the AUTO_INCREMENT sequence start with another value, using the following SQL statement

```
ALTER TABLE Customers AUTO_INCREMENT=100;
```

- See the SQL documentation for whatever database you are using for the auto-increment field syntax.