

Chapter 25 Java Database Programming

1. A *superkey* is an attribute or a set of attributes that uniquely identifies the relation. That is, no two tuples have the same values on the superkey. A key \underline{K} is a minimal superkey, meaning that any proper subset of \underline{K} is not a superkey. A relation can have several keys. In this case, each of the keys is called a *candidate key*. The *primary key* is one of the candidate keys designated by the database designer. The primary key is often used to identify tuples in a relation. You may specify a primary in the create table statement using the primary key clause.
2. A set of attributes FK is a *foreign key* in a relation R that references relation T if it satisfies the following two rules:

- The attributes in FK have the same domain as the primary key in T .
- A non-null value on FK in R must match a primary key value in T .

You may specify a foreign key in the create table statement using the foreign key clause.

3. A relation can have only one primary key, but may have multiple foreign keys.
4. No.
5. No, but they must have the same domain.
6. Yes.
7. See www.cs.armstrong.edu/intro5e/ch26tables.sql
- 8.

```
select * from Course where subjectId = 'CSCI' and
numOfCredits >= 4
```

9.

```
select * from Student where lastName like '%S%S%'
```

10.

```
select * from Student where birthdate is null
```

11.

```
select distinct firstName, lastName from Student, Enrollment,  
Course where Student.ssn = Enrollment.ssn and Course.courseId =  
Enrollment.courseId
```

12.

```
select subjectId, count(*) from Course group by subjectId
```

13.

```
select Student.ssn, 50 * sum(numOfCredits) from Student,  
Enrollment, Course where Student.ssn = Enrollment.ssn and  
Course.courseId = Enrollment.courseId group by Student.ssn
```

14.

(1) platform independence, i.e., your Java program can run on any platform and access any relational database. (2) Java has an extensive set of classes and interfaces in the API that you can use to develop database applications and applets productively and efficiently.

15. A JDBC application loads an appropriate driver using the Driver interface, connects to the database using the Connection interface, creates and executes SQL statements using the Statement interface, and processes the result using the ResultSet interface if the statements return results.

16. Use the `Class.forName(driverName)` method to load the driver with its full name. The driver class for MySQL, Access, and Oracle are `com.mysql.jdbc.Driver`, `sun.jdbc.odbc.JdbcOdbcDriver`, `oracle.jdbc.driver.OracleDriver`, respectively.

17. To create a JDBC connection, use `DriverManager.getConnection(url)`. The URLs for MySQL, Access, and Oracle are
`jdbc:mysql://liang.armstrong.edu/test`
`jdbc:odbc:exampleMDBDataSource"`,
`jdbc:oracle:thin:@liang.armstrong.edu:1521:ora9i`.

18. To create an instance of Statement, use `connection.createStatement()`. To execute a statement, use the methods `executeQuery(...)` and `executeUpdate(...)`. `executeQuery(...)` returns a result set, but `executeUpdate(...)` does not return a result set.

19. To retrieve values in a ResultSet, use `next()` to move the cursor to the next row and use the `getXxx(number)` or `getXxx(columnName)` method to retrieve fields from the current row.

21. JDBC automatically commits a transaction. You can set `autoCommit` to false using `setAutoCommit(false)` on a `Connection` object.
20. `PreparedStatement` is a subinterface of `Statement`. To create a `PreparedStatement`, use `connection.prepareStatement(String sql)`, where `sql` is a prepared statement with parameters denoted using question marks.
22. Since the prepared statements are precompiled, they are efficient for repeated executions. To execute a `PreparedStatement`, first set parameter values using the `setX(int parameterIndex, X value)` method, then invoke `execute()` method.
23. The `DatabaseMetaData` interface contains the methods for obtaining database-wide information.

The general information includes the URL, username, product name, product version, driver name, driver version, available functions, available data types, and so on. The methods for general information usually return a string, an integer, except that the method for retrieving available data types returns a `ResultSet`. Most methods of this type don't have parameters.

Information on database capabilities includes such matters as whether the database supports the GROUP BY operator, the ALTER TABLE command with add column option, and entry-level or full ANSI92 SQL grammar. The methods for finding database capabilities return a `boolean` value indicating whether the database possesses a certain capability. Most of methods of this type don't have parameters and are named with prefix *supports*. Table 13.3 gives some of.

The methods for getting database objects return lists of information in `ResultSets`. You can use the normal `ResultSet` methods, such as `getString` and `getInt`, to retrieve data from these `ResultSets`. If a given form of metadata is not available, these methods should throw a `SQLException`.

To create an instance of `DatabaseMetaData`, use the `getDatabaseMetaData()` method from a `Connection` object.

24. The `ResultSetMetaData` interface describes information pertaining to the result set. A `ResultSetMetaData` object can be used to find out about the types and properties of the columns in a `ResultSet`. The methods in `ResultSetMetaData` have a single `int` parameter representing the column except that the `getColumnCount` method has no parameters. All these methods return `int`, `boolean`, or `String`. To create an instance of `ResultSetMetaData`, use `getResultSetMetaData` from an instance of `ResultSet`.

25. To find the number of columns in a result set, first create an instance of `ResultSetMetaData` using the `getMetaData()` method on a `ResultSet`. Then use `getColumnCount()` to return the column count and use `getColumnName(int)` to return the column name.

26. Batch processing enables SQL nonselect statements to be processed all together. Therefore, it is more efficient than processing each statement individually.

27. To add an SQL statement to a batch, you add it to a `Statement` object using the `addBatch(sqlString)` method. Use `executeBatch()` method to execute it.

28. No, you cannot execute a `SELECT` statement in a batch.

29.

To find out whether a driver supports batch updates, invoke `supportsBatchUpdates()` on a `DatabaseMetaData` instance. If the driver supports batch updates, it will return true. The JDBC drivers for MySQL, Access, and Oracle all support batch updates.

30. A scrollable result set allows you to move the cursor in any order and place it anywhere. By default, a result set is non-scrollable and you can only traverse it forward from beginning to end. A scrollable result set may be updateable, i.e., insert, delete, update rows in the table through a result set.

31.

You use the following statement to create a scrollable and updateable result set:

```
Statement statement = connection.createStatementResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_UPDATABLE)
```

32.

You can use `supportsResultSetType(int type)` and `supportsResultSetConcurrency(int type, int concurrency)` in the `DatabaseMetaData` interface to find out which result type and concurrency modes are supported by the JDBC driver.

33.

You have to use the SQL3 new data type `BLOB` to declare a column in the table, and use JDBC to prepared statement to prepare an `INSERT` statement. To set the data to the field, use the `setBinaryStream` method.

34.

To retrieve it, use the getBlob() method to obtain the binary data in the Blob type and use the getBytes method to return an array of bytes from the Blob value. To create an image, use ImageIcon(byte[]).

35.

Oracle supports SQL3 BLOB and CLOB types. MySQL support CLOB. Access supports none.