

A Relational Database Overview

A database is a means of storing information in such a way that information can be retrieved from it. In simplest terms, a relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database. A Database Management System (DBMS) handles the way data is stored, maintained, and retrieved. In the case of a relational database, a Relational Database Management System (RDBMS) performs these tasks. DBMS as used in this book is a general term that includes RDBMS.

Integrity Rules

Relational tables follow certain integrity rules to ensure that the data they contain stay accurate and are always accessible. First, the rows in a relational table should all be distinct. If there are duplicate rows, there can be problems resolving which of two possible selections is the correct one. For most DBMSs, the user can specify that duplicate rows are not allowed, and if that is done, the DBMS will prevent the addition of any rows that duplicate an existing row.

A second integrity rule of the traditional relational model is that column values must not be repeating groups or arrays. A third aspect of data integrity involves the concept of a null value. A database takes care of situations where data may not be available by using a null value to indicate that a value is missing. It does not equate to a blank or zero. A blank is considered equal to another blank, a zero is equal to another zero, but two null values are not considered equal.

When each row in a table is different, it is possible to use one or more columns to identify a particular row. This unique column or group of columns is called a primary key. Any column that is part of a primary key cannot be null; if it were, the primary key containing it would no longer be a complete identifier. This rule is referred to as entity integrity.

The `Employees` table illustrates some of these relational database concepts. It has five columns and six rows, with each row representing a different employee.

Employees Table

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

The primary key for this table would generally be the employee number because each one is guaranteed to be different. (A number is also more efficient than a string for making comparisons.) It would also be possible to use `First_Name` and `Last_Name` because the combination of the two also identifies just one row in our sample database. Using the last name alone would not work because there are two employees with the last name of "Washington." In this particular case the first names are all different, so one could conceivably use that column as a primary key, but it is best to avoid using a column where duplicates could occur. If Elizabeth Jones gets a job at this company and the primary key is `First_Name`, the RDBMS will not allow her name to be added (if it has been specified that no duplicates are permitted). Because there is already an Elizabeth in the table, adding a second one would make the primary key useless as a way of identifying just one row. Note that although using `First_Name` and `Last_Name` is a unique composite key for this example, it might not be unique in a larger database. Note also that the `Employees` table assumes that there can be only one car per employee.

SELECT Statements

SQL is a language designed to be used with relational databases. There is a set of basic SQL commands that is considered standard and is used by all RDBMSs. For example, all RDBMSs use the `SELECT` statement.

A `SELECT` statement, also called a query, is used to get information from a table. It specifies one or more column headings, one or more tables from which to select, and some criteria for selection. The RDBMS returns rows of the column entries that satisfy the stated requirements. A `SELECT` statement such as the following will fetch the first and last names of employees who have company cars:

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number IS NOT NULL
```

The result set (the set of rows that satisfy the requirement of not having null in the `Car_Number` column) follows. The first name and last name are printed for each row that satisfies the requirement because the `SELECT` statement (the first line) specifies the columns `First_Name` and `Last_Name`. The `FROM` clause (the second line) gives the table from which the columns will be selected.

FIRST_NAME	LAST_NAME
John	Washington
Florence	Wojokowski

The following code produces a result set that includes the whole table because it asks for all of the columns in the table `Employees` with no restrictions (no `WHERE` clause). Note that `SELECT *` means "SELECT all columns."

```
SELECT *
FROM Employees
```

WHERE Clauses

The `WHERE` clause in a `SELECT` statement provides the criteria for selecting values. For example, in the following code fragment, values will be selected only if they occur in a row in which the column `Last_Name` begins with the string 'Washington'.

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Last_Name LIKE 'Washington%'
```

The keyword `LIKE` is used to compare strings, and it offers the feature that patterns containing wildcards can be used. For example, in the code fragment above, there is a percent sign (%) at the end of 'Washington', which signifies that any value containing the string 'Washington' plus zero or more additional characters will satisfy this selection criterion. So 'Washington' or 'Washingtonian' would be matches, but 'Washing' would not be. The other wildcard used in `LIKE` clauses is an underbar (_), which stands for any one character. For example,

```
WHERE Last_Name LIKE 'Ba_man'
```

would match 'Barman', 'Badman', 'Balman', 'Bagman', 'Bamman', and so on.

The code fragment below has a `WHERE` clause that uses the equal sign (=) to compare numbers. It selects the first and last name of the employee who is assigned car 12.

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number = 12
```

The next code fragment selects the first and last names of employees whose employee number is greater than 10005:

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number > 10005
```

`WHERE` clauses can get rather elaborate, with multiple conditions and, in some DBMSs, nested conditions. This overview will not cover complicated `WHERE` clauses, but the following code fragment has a `WHERE` clause with two conditions; this query selects the first and last names of employees whose employee number is less than 10100 and who do not have a company car.

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number < 10100 and Car_Number IS NULL
```

A special type of `WHERE` clause involves a join, which is explained in the next section.

Joins

A distinguishing feature of relational databases is that it is possible to get data from more than one table in what is called a join. Suppose that after retrieving the names of employees who have company cars, one wanted to find out who has which car, including the license plate number, mileage, and year of car. This information is stored in another table, `Cars`:

Cars Table

Car_Number	License_Plate	Mileage	Year
------------	---------------	---------	------

5	ABC123	5000	1996
12	DEF123	7500	1999

There must be one column that appears in both tables in order to relate them to each other. This column, which must be the primary key in one table, is called the foreign key in the other table. In this case, the column that appears in two tables is `Car_Number`, which is the primary key for the table `Cars` and the foreign key in the table `Employees`. If the 1996 car with license plate number ABC123 were wrecked and deleted from the `Cars` table, then `Car_Number` 5 would also have to be removed from the `Employees` table in order to maintain what is called referential integrity. Otherwise, the foreign key column (`Car_Number`) in the `Employees` table would contain an entry that did not refer to anything in the `Cars` table. A foreign key must either be null or equal to an existing primary key value of the table to which it refers. This is different from a primary key, which may not be null. There are several null values in the `Car_Number` column in the table `Employees` because it is possible for an employee not to have a company car.

The following code asks for the first and last names of employees who have company cars and for the license plate number, mileage, and year of those cars. Note that the `FROM` clause lists both the `Employees` and `Cars` tables because the requested data is contained in both tables. Using the table name and a dot (.) before the column name indicates which table contains the column.

```
SELECT Employees.First_Name, Employees.Last_Name,
       Cars.License_Plate, Cars.Mileage, Cars.Year
FROM Employees, Cars
WHERE Employees.Car_Number = Cars.Car_Number
```

This returns a result set that will look similar to the following:

FIRST_NAME	LAST_NAME	LICENSE_PLATE	MILEAGE	YEAR
John	Washington	ABC123	5000	1996
Florence	Wojokowski	DEF123	7500	1999

Common SQL Commands

SQL commands are divided into categories, the two main ones being Data Manipulation Language (DML) commands and Data Definition Language (DDL) commands. DML commands deal with data, either retrieving it or modifying it to keep it up-to-date. DDL commands create or change tables and other database objects such as views and indexes.

A list of the more common DML commands follows:

- `SELECT` — used to query and display data from a database. The `SELECT` statement specifies which columns to include in the result set. The vast majority of the SQL commands used in applications are `SELECT` statements.
- `INSERT` — adds new rows to a table. `INSERT` is used to populate a newly created table or to add a new row (or rows) to an already-existing table.
- `DELETE` — removes a specified row or set of rows from a table
- `UPDATE` — changes an existing value in a column or group of columns in a table

The more common DDL commands follow:

- `CREATE TABLE` — creates a table with the column names the user provides. The user also needs to specify a type for the data in each column. Data types vary from one RDBMS to another, so a user might need to use metadata to establish the data types used by a particular database. `CREATE TABLE` is normally used less often than the data manipulation commands because a table is created only once, whereas adding or deleting rows or changing individual values generally occurs more frequently.
- `DROP TABLE` — deletes all rows and removes the table definition from the database. A JDBC API implementation is required to support the `DROP TABLE` command as specified by SQL92, Transitional Level. However, support for the `CASCADE` and `RESTRICT` options of `DROP TABLE` is optional. In addition, the behavior of `DROP TABLE` is implementation-defined when there are views or integrity constraints defined that reference the table being dropped.
- `ALTER TABLE` — adds or removes a column from a table. It also adds or drops table constraints and alters column attributes

Result Sets and Cursors

The rows that satisfy the conditions of a query are called the result set. The number of rows returned in a result set can be zero, one, or many. A user can access the data in a result set one row at a time, and a cursor provides the means to do that. A cursor can be thought of as a pointer into a file that contains the rows of the result set, and that pointer has the ability to keep track of which row is currently being accessed. A cursor allows a user to process each row of a result set from top to bottom and consequently may be used for iterative processing. Most DBMSs create a cursor automatically when a result set is generated.

Earlier JDBC API versions added new capabilities for a result set's cursor, allowing it to move both forward and backward and also allowing it to move to a specified row or to a row whose position is relative to another row.

See [Retrieving and Modifying Values from Result Sets](#) for more information.

Transactions

When one user is accessing data in a database, another user may be accessing the same data at the same time. If, for instance, the first user is updating some columns in a table at the same time the second user is selecting columns from that same table, it is possible for the second user to get partly old data and partly updated data. For this reason, DBMSs use transactions to maintain data in a consistent state (data consistency) while allowing more than one user to access a database at the same time (data concurrency).

A transaction is a set of one or more SQL statements that make up a logical unit of work. A transaction ends with either a commit or a rollback, depending on whether there are any problems with data consistency or data concurrency. The commit statement makes permanent the changes resulting from the SQL statements in the transaction, and the rollback statement undoes all changes resulting from the SQL statements in the transaction.

A lock is a mechanism that prohibits two transactions from manipulating the same data at the same time. For example, a table lock prevents a table from being dropped if there is an uncommitted transaction on that table. In some DBMSs, a table lock also locks all of the rows in a table. A row lock prevents two transactions from modifying the same row, or it prevents one transaction from selecting a row while another transaction is still modifying it.

See [Using Transactions](#) for more information.

Stored Procedures

A stored procedure is a group of SQL statements that can be called by name. In other words, it is executable code, a mini-program, that performs a particular task that can be invoked the same way one can call a function or method. Traditionally, stored procedures have been written in a DBMS-specific programming language. The latest generation of database products allows stored procedures to be written using the Java programming language and the JDBC API. Stored procedures written in the Java programming language are bytecode portable between DBMSs. Once a stored procedure is written, it can be used and reused because a DBMS that supports stored procedures will, as its name implies, store it in the database. See [Using Stored Procedures](#) for information about writing stored procedures.

Metadata

Databases store user data, and they also store information about the database itself. Most DBMSs have a set of system tables, which list tables in the database, column names in each table, primary keys, foreign keys, stored procedures, and so forth. Each DBMS has its own functions for getting information about table layouts and database features. JDBC provides the interface `DatabaseMetaData`, which a driver writer must implement so that its methods return information about the driver and/or DBMS for which the driver is written. For example, a large number of methods return whether or not the driver supports a particular functionality. This interface gives users and tools a standardized way to get metadata. In general, developers writing tools and drivers are the ones most likely to be concerned with metadata.