Databases

Contents

Definitions

An **entity** is a category of object, person, event, or thing about which data needs to be recorded.

A simple (flat file) database is a single file containing information on a single entity.

An entity maps to a table. A table contains many records, each holding information about a particular instance of an entity. For example, a `tblCustomer` table will contain many records, each representing a single customer.

Each record of an entity table needs an identifier to uniquely identify it. The identifier is known as the **primary key**.

A composite primary key is a combination of keys in a record, where the *combination* uniquely identifies the record rather than having a separate ID key.

Most databases automatically index the primary key field so that any particular record can be found quickly.

If another key field also needs to be searched for, then it can be called a **secondary key**, and this field will be indexed.

Entities can be linked by one-to-one, one-to-many, or many-to-many relationships. See the diagrams.

To create a relationship between entities, the ‘many’ side must have a **foreign key**, which matches the primary key in another table. Say a customer can have many subscriptions. Each record must have a fixed number of fields, so they can’t have an arbitrary number of foreign keys to link to subscriptions. Therefore, each subscription record should have a foreign key to point to a customer.

Referential integrity means that every foreign key must reference a existent record in a related table that actually exists. Dangling foreign keys are forbidden.

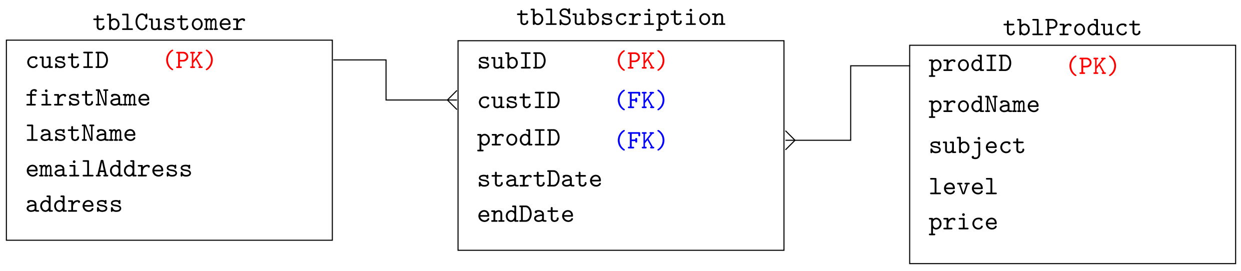
As a canonical example, a customer can have many orders; an order can have many orderLines (an order is like a shopping cart, and an orderLine is like a single order with product, quantity, price); a product can appear in many orderLines.

Many-to-many relationships cause problems with unknowable column numbers. This can be fixed with a **linking table**. Consider a fitness center - members can take many classes, and classes can have many members. Instead, have an enrolment table in the middle. A member can have many enrolments, and a class can be part of many enrolments. Each enrolment record is a single link between a member and a class.

Entity-Relationship Diagrams

Boxes contain entities, or the fields of a record in that table. Relationship lines have a single line connection to the box for a *one* connection, and a crow's foot connection for *many*.

In this example, there is an online revision service which provides textbooks. A customer can subscribe to different tiers. A customer has personal details; a product is a book with a name, subject, level, and price; and a subscription links these two together with foreign keys for each, as well as a start and end date.



The `tblCustomer` table has personal details and a unique ID, which is its primary key. The `tblProduct` table has details about the product and its own unique ID primary key. The `tblSubscription` table has a unique ID primary key, a start and end date, and two foreign keys. Each subscription references the customer that subscribes to it and the product they subscribe to, in the form of two primary keys.

Normalisation

Normalisation is a process used to come up with the best possible design for a database. Tables should be organised so that data is not duplicated in the same table or different tables. The structure should allow complex queries to be made.

A normalised database is much easier to maintain and change; there is no unnecessary duplication of data; integrity is maintained and updates are small; having smaller tables with fewer fields makes searching faster and saves storage.

There are 3 main stages of normalisation:

First Normal Form (1NF)

A table is 1NF normalised if it contains no repeating attributes or groups of attributes. **All attributes must be atomic (separate first and last names etc.)**. A field cannot be a composite data type. It must instead by a foreign key referencing another table which contains the desired data.

Second Normal Form (2NF)

A table is 2NF if it's 1NF and **contains no partial dependencies**.

Identify a table's purpose. Each column must describe the singular thing that the primary key identifies. If it doesn't, then it should belong in a different table. For example, in a `tblEmployee`, all columns should describe an attribute of the particular employee identified by the primary key.

If a table has a composite primary key, then it is only in 2NF if each column directly relates to *both* parts of the primary key. The table must refer entirely to a singular entity.

Third Normal Form (3NF)

A table is 3NF if it's 2NF and **contains no non-key dependencies**. All attributes are dependent entirely on the primary key and nothing else. This means that the primary key determines all other attributes in its record.

SQL

SQL is a declarative language used for querying and updating tables in a relational database.

The `SELECT` statement is used to extract fields from one or more tables. The syntax is the following:

```
SELECT (fields) FROM (tables) [WHERE (criteria)] [SELECTORS].. [ORDER BY (field) [ASC | DESC]];
```

An example would be:

```
SELECT (prodID, name, subject, price) FROM tblProduct WHERE level = 4 AND price BETWEEN 500 AND 1500
ORDER BY name;
```

Every SQL statement **MUST** end in a semicolon. A logical statement can span multiple physical lines, so the semicolon is needed to end the statement.

`WHERE` clauses

Wild cards can be used, and act like they do in glob patterns. `*` matches anything. `LIKE` can be used in a `WHERE` clause to find fields by matching a pattern.

`WHERE` clauses can also accept ranges with `BETWEEN` or `IN`. The `BETWEEN` selector finds values in an inclusive numerical range. `IN` finds values where they match one of several possibilities.

All standard mathematical comparison operators can be used, as well as `AND`, `OR`, as well as `NOT`.

Using multiple tables

Multiple tables can be selected from using a `SELECT` statement. This is most often useful for a `WHERE` clause. Consider this expression:

```
SELECT tblCustomer.firstName, tblCustomer.lastName, tblProduct.name
  FROM tblCustomer, tblProduct, tblSubscription
  WHERE tblSubscription.custID = tblCustomer.custID
  AND tblSubscription.prodID = tblProduct.prodID;
```

This will select the customer name and product name for every subscription. There will be one row for each subscription and each row will contain the appropriate customer and product. This query can be made simpler by using the `JOIN` clause.

`JOIN`-ing tables

Data can be extracted using the join clause. You can select from multiple tables and join them together by their foreign key relationships. This then treats the results as one table, which you can then use something like a `WHERE` clause with.

Here is a `SELECT` statement equivalent to the previous one, but using `JOIN` clauses instead:

```
SELECT tblCustomer.firstName, tblCustomer.lastName, tblProduct.name
  FROM tblSubscription
  JOIN tblCustomer ON tblCustomer.custID = tblSubscription.custID
  JOIN tblProduct ON tblProduct.prodID = tblSubscription.prodID;
```

The `JOIN` clause is used to match foreign keys between tables to search them together.

Defining and updating tables with SQL

Creating tables

The `CREATE TABLE` command expects a table name and a comma-separated list of fields. Each field has a name, a type, and some extra information about the field, such as `NOT NULL` or `PRIMARY KEY`. For example:

```
CREATE TABLE tblProduct (
    productID   CHAR(4) NOT NULL UNIQUE PRIMARY KEY,
    description VARCHAR(20) NOT NULL,
    price       CURRENCY
);
```

Data types

`CHAR(`n`)`	String of fixed length n
`VARCHAR(`n`)`	Variable length string of max length n
`BOOLEAN`	True or false
`INTEGER` or `INT`	Integer
`FLOAT`	Floating point decimal number
`DATE`	Date in local format
`TIME`	Time in local format
`CURRENCY`	Currency in local format

Defining links between tables

When defining a new table with foreign keys, we want to ensure that foreign keys always reference valid keys in other tables. We also want to be able to specify composite primary keys. These things can both be done by adding extra lines to the initial table definition.

For example, to link products and components with a many-to-many relationship, you can use a linking table in the middle like so:

```
CREATE TABLE productComponent (
    productID   CHAR(4) NOT NULL,
    componentID CHAR(6) NOT NULL,
    quantity    INTEGER,
    FOREIGN KEY productID REFERENCES tblProduct(productID),
    FOREIGN KEY componentID REFERENCES tblComponent(componentID),
    PRIMARY KEY (productID, componentID)
);
```

Altering a table

The `ALTER TABLE` command is used to alter tables by adding, deleting, or modifying the columns of the table. Some examples:

```
ALTER TABLE tblProduct ADD qtyInStock INTEGER;
ALTER TABLE tblCustomer DROP COLUMN postcode;
ALTER TABLE tblProduct MODIFY COLUMN productName NOT NULL;
```

Inserting, updating, and deleting data

The `INSERT INTO` command is used to insert a new record into a table. For example:

```
INSERT INTO tblCustomer (customerID, firstName, lastName, dateOfBirth, email)
  VALUES ("JS3592", "John", "Smith", DATE "1982-04-16", "johnnyboysmith@gmail.com");
```

The `UPDATE` command can update a set of records (possibly a set of size 1, meaning a single record) and change fields of the selected records. For example:

```
UPDATE tblCustomer
  SET email = "johnsmith1982@gmail.com"
  WHERE customerID = "JS3592";
```

The `DROP` command is used to delete a whole table or a column of it, but `DELETE` is used to delete individual records from a table. For exmaple:

```
DELETE FROM tblCustomer
  WHERE customerID = "JS3592";
```