Database Systems Laboratory work 1

Week 2: Relational models and keys

Task 1.1

BookID	ISBN	Title	Author	Publisher	Year	ShelfNo
1	978-032157	Database Systems	Elmasri	Pearson	2016	A12
2	978-013468	Java Programming	Deitel	Pearson	2018	B07
3	978-026203	Algorithms	Cormen	MIT Press	2009	C22

Superkeys:

- {BookID}
- {ISBN}
- {BookID, ISBN}
- {BookID, Title, Year}
- {ISBN, ShelfNo, Year}

Candidate keys:

- {BookID} (unique for each book)
- {ISBN}(unique number for identifying books)

Primary key (BookId), BookID is guaranteed to be unique, non-null, and never changes, while ISBN is also unique, but it is a natural key from an external system and could change.

Yes, two books can have the same title, This is why "Title" alone cannot be a candidate key.

Task 1.2

Given table: BookLoan(StudentID, BookID, LoanDate, DueDate, ReturnDate, FineAmount)

Primary key {StudentID, BookID, LoanDate}

StudentID is needed to identify which student made the loan. Samely, BookID is needed to identify which book is loaned. Since a student can borrow the same book multiple times but not in one date, LoanDate is necessary.

A potential additional candidate key could be a surrogate key like LoanID. If a unique LoanID was assiigned to every loan event, it would also uniquely identify each tuple and could be chosen as a simple primary key. The combination {StudentID, BookID, LoanDate} would remain a candidate key.

Task 1.3

• Student (StudentID, Name, Email, Major)

• Book (BookID, ISBN, Title, Author, Publisher, Year)

• Loan(LoanID, StudentID, BookID, LoanDate, ReturnDate)

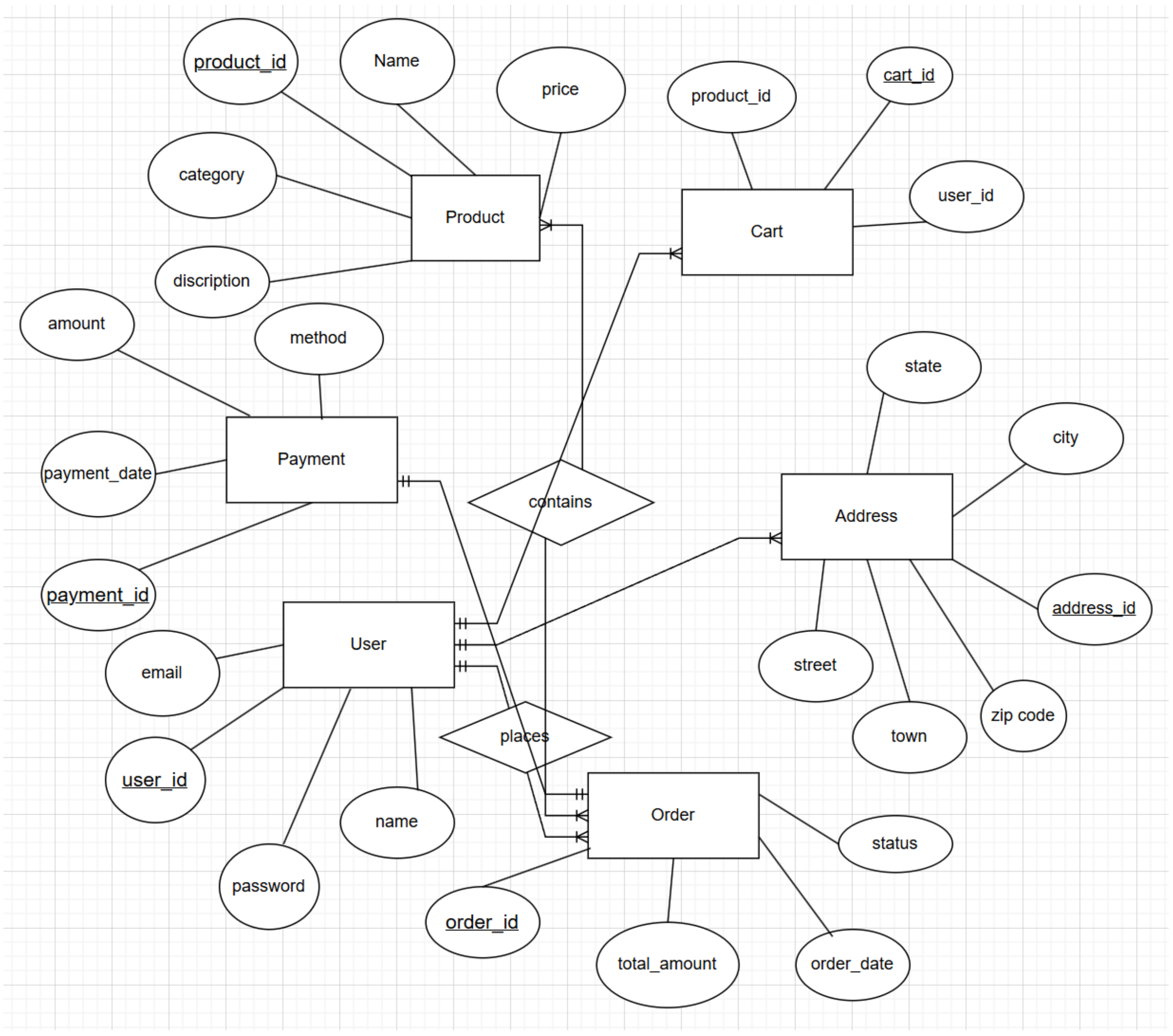
• Librarian(LibID, Name, Email, Office)

Foreign keys: StudentID, BookID are foreign keys in table Loan since they are primary keys in Student and Book tables, respectively.

StudentID.Loan -> StudentID.Student, this ensures that every loan record is associated with a valid, existing student.

BookID.Loan -> BookID.Book, this ensures that everu loan record is associated witha valid, existing book.

Task 2



Task 3.1

Functional dependencies:

Employee_id -> name, city_code, home_city

Job_code -> job

City_code -> home_city

The primary key must be a composite key: {employee_id, job_code} because an employee can have multiple jobs

Redundancies: name, city_code, home_city are repeated for every job an employee has

Anomalies: update anomaly: if someone moves to another city we must update his city_code and home_city

Insertion anomaly: we cannot add a new city until an employee is assigned to live there

Deletion anomaly: if we delete all employees who are waiters (job_code - J02), we lose the information that J02 corresponds to the job "Waiter"

1NF – the table is already in that form because all attributes are atomic

2NF (remove partial dependencies). A partial dependency exists because name, city_code, and home_city depend only on part of the primary key(employee_id)

Employees (employee_id, name, city_code, home_city),

Jobs (employee_id, job_code, job)

3NF (remove transitive dependencies). In the Employees table, home_city is transitively dependent on the primary key via city_code

Employees (employee_id, name, city_code),

Cities (city_code, home_city),

Assignments (employee_id, job_code),

JobCodes (job_code, job)

Task 3.2

UniversityEnrollment(Author, Nationality, Book title, Genre, Number of pages)

Primary key: {Author, Book title}

Functional dependencies: Author -> Nationality

Book title -> Genre, Number of pages

The table is not in BCNF because BCNF requires that for every non-trivial functional dependency A -> B, A must be a superkey.

Final BCNF:

Authors (Author, Nationality)

Books (Book Title, Genre, Number of pages)

BookAuthors (Author, Book Title)

If a natural join was performed on BookAuthors, Authors, and Books on the correct attributes, the original table will be recontracted without any spurious tuples, so no information is lost.