

Functional Dependencies

Dr. Katharina Schneider & Dr. Tony Garnock-Jones
BCS1510

EPD150 MSM Conference Hall
29 April 2025

Review of Part I

1. Course introduction ✓
2. SQL: Relational model, **DDL**, **DQL**, **DML** ✓
 - Queries, subqueries, operators, set operators, ...
3. Advanced subqueries, aggregations (GROUP BY, HAVING)... ✓
4. Data modification and manipulation ✓
 - INSERT, DELETE, UPDATE, defaults, conditional operations...
5. Built-in functions ✓
6. Views, Constraints, Triggers, COALESCE ✓

Part II

Design Theory for Relational Databases

Outline of Part II

1. **Today:** Functional Dependencies & Keys
2. Normal Forms
3. Query Tuning
4. Transactions; Beyond the Relational Model

Learning Objectives

Today we focus on **relational schema design**, in order to minimize *redundancy* and avoid *update & deletion anomalies* in our databases.

1. Understand Functional Dependencies (FDs)
2. Identify, define, and reason using keys and superkeys
3. Derive keys to relations from Functional Dependencies

Our Running Example

Most of our SQL queries will be based on the following database schema. Underline = primary key attributes, as always!

- Products(maker, model, type)
- PCs(model, speed, ram, hd, price)
- Laptops(model, speed, ram, hd, screen, price)
- Printers(model, color, type, price)
- Customers(customer_id, firstname, lastname, city, address, email)
- Sales(customer_id, model, quantity, day, paid, type_of_payment)

Additional Assumptions

Not reflected in the schema, but nonetheless true of our database:

- Each customer has a **unique** email address.
 - Another way of saying this: no two customers share an email address.
- We want to always have the **latest address** for a customer.
 - If they tell us a new address, we will update our records.

Functional Dependencies

Alternative [less-smart] Schema

- Replace:
 - Products(maker, model, type)
 - Customers(customer_id, firstname, lastname, city, address, email)
 - Sales(customer_id, model, quantity, day, paid, type_of_payment)
- with:
 - Productsales(customer_id, firstname, lastname, city, address, email, model, quantity, day, paid, type_of_payment, maker, type)

Is having Productsales worse than Customers and Sales separately? **What could be wrong about Productsales?**

It's redundant, repetitive, repetitious, and redundant!

- Every distinct **customer_id** should always have the same **city, firstname, lastname, ..., email, ...**
- Every distinct *non-NULL* **email** should always have the same **customer_id**
- Every distinct **model** should always have the same **maker** and **type**
- Every distinct (**customer_id, model, day**) should always have the same **quantity, paid, type_of_payment**

Why do we make these claims, though?

Idea: Functional Dependencies (FDs)

Each subset of the attributes that *completely determines* another subset is called a **Functional Dependency (FD)**.

customer_id \rightarrow *firstname lastname city address email*

~~*email* \rightarrow *customer_id*~~

model \rightarrow *maker type*

customer_id model day \rightarrow *quantity paid type_of_payment*

Idea: Functional Dependencies (FDs)

Consider these columns of Productsales (not all columns, because of space limitations) and the following data:

customer_id	city	model	maker	quantity	day	paid
9999999999	Limerick	1007	C	1	2013-12-20	459.00
9999999999	Limerick	3007	H	1	2013-12-20	360.00
9876543210	Galway	1007	C	3	2013-12-19	1530.00
1122334455	Dublin	2010	G	1	2013-12-19	2300.00

customer_id → city

model → maker


customer_id model day → quantity

customer_id model day → paid

Relational Schema Design

Goal of relational schema design is to *avoid redundancy* of data and minimise the risk of *anomalies*.


Update anomaly: we change one occurrence of a fact in the database, but other occurrences of the same fact remain unchanged. **Example:**

- Update city for customer 9999999999 in the first row in Productsales
- ... and forget to change it in the second. 

Relational Schema Design

Goal of relational schema design is to *avoid redundancy* of data and minimise the risk of *anomalies*.

Update anomaly: we **insert a new** occurrence of a fact in the database, but other occurrences of the same fact remain unchanged. **Example:**

- Add a new sale for customer 9999999999 in Productsales, with a new city
- ... but none of the other rows change! 

Relational Schema Design

Goal of relational schema design is to *avoid redundancy* of data and minimise the risk of *anomalies*.

Deletion anomaly: by deleting one fact from the database we unintentionally delete other facts as well. **Example:**

- Delete the fact we sold model 2010 to customer 1122334455
- ... and we lose the facts that 2010 is a laptop and customer 1122334455 is from Dublin. 🤪

Functional Dependencies (FDs)

Formally

Consider relation **R** and let X and Y be two sets of attributes of **R**. We say that $X \rightarrow Y$ (X functionally determines Y) in **R** if:

- when two tuples of **R** have the same values for all the attributes in X ,
- then they also have the same values for all attributes in Y .

| Let $r, s \in R$. If $\pi_{(X)}(\{r\}) = \pi_{(X)}(\{s\})$, then $\pi_{(Y)}(\{r\}) = \pi_{(Y)}(\{s\})$.

Functional Dependencies (FDs)

Formally

We say that a relation **R** *satisfies* a functional dependency if it holds for all **conceivable** instances of **R**.

<i>title</i>	<i>year</i>	<i>length</i>	<i>genre</i>	<i>studioName</i>	<i>starName</i>
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure 3.2: An instance of the relation `Movies1(title, year, length, genre, studioName, starName)`

Functional Dependencies (FDs)

Notational conventions

X, Y, Z represent **sets** of attributes; A, B, C represent **single** attributes.

We don't write the braces $\{...\}$ in sets of attributes; just ABC , rather than $\{A, B, C\}$.

Functional Dependencies (FDs)

In terms of relational algebra

If

$$X \rightarrow Y \text{ in } \mathbf{R}$$

then

$$|\pi_{(X)}(\mathbf{R})| = |\pi_{(X \cup Y)}(\mathbf{R})|$$

Exercise: why is this? Write down a proof.

Functional Dependencies (FDs)

As functions projected from a relation

If

$$A_1, \dots, A_n \rightarrow B_1, \dots, B_m \text{ in } \mathbf{R}$$

then

$$\begin{aligned} f(a_1, \dots, a_n) &= (b_1, \dots, b_m) \\ \text{where } (a_1, \dots, a_n, b_1, \dots, b_m) \\ &\in \pi_{(A_1, \dots, A_n, B_1, \dots, B_m)}(\mathbf{R}) \end{aligned}$$

is a function.

Figure 3.1 suggests what this FD tells us about any two tuples t and u in the relation R . However, the A 's and B 's can be anywhere; it is not necessary for the A 's and B 's to appear consecutively or for the A 's to precede the B 's.

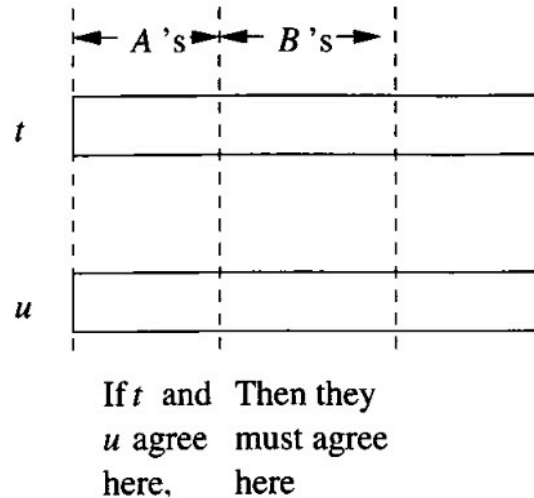


Figure 3.1: The effect of a functional dependency on two tuples.

Some Observations: Trivial FDs

- Unique attributes (including, but not limited to, primary keys) always determine all other attributes.
- **Trivial** FDs:
 - Let X be a set of attributes; let $A \in X$. Then $X \rightarrow A$.
 - (Corollary:) Let Z be the set of all attributes of \mathbf{R} . Then $Z \rightarrow A$ for any attribute A of \mathbf{R} .

Some Observations: Repetition

If $X \rightarrow Y$ in \mathbf{R} and two *distinct* rows exist in \mathbf{R} with the same values for the X s, then the Y s will also be repeated.

This repetition is unwanted if we want to store each fact only once in the database.

Example: $customer_id \rightarrow city$ in Productsales. If the same customer is present in multiple rows, we will have the city of that customer repeated multiple times.

→ **Risk of update anomaly!**

Some Observations: Many ways of reading

Let $XY \rightarrow A$ and $Y \rightarrow A$ in **R**. If we delete a row based on the values of X and Y together, we will also delete a combination of values of Y and A which is an independent fact from the fact represented by the combination of X and Y .

Example: In Productsales,

- *customer_id model day* \rightarrow *city* and
- *customer_id* \rightarrow *city*

If we delete the fact that customer 1122334455 bought model 2010 on 2013-12-19, then we also forget they live in Dublin!

→ **Risk of deletion anomaly!**

Splitting Right Sides of FDs

$X \rightarrow A_1, A_2, \dots, A_n$ in **R** is equivalent to

- $X \rightarrow A_1$
- $X \rightarrow A_2$
- ...
- $X \rightarrow A_n$

Example: *customer_id model day \rightarrow quantity paid \Leftrightarrow*

- *customer_id model day \rightarrow quantity*
- *customer_id model day \rightarrow paid*

There is no splitting rule for left sides.

Who decides FDs? Why?

FDs are *asserted* by **whoever designs the database schema**. They are then "*true by definition*" in that schema.

FDs are a good indication for the **risk of anomalies** in a relational database schema... so:

→ *Informal* goal of relational database schema design:

Have as few FDs with *non-unique* LHS as possible.

Once we have some FDs, we can **deduce other** FDs and work on **minimising** the set of FDs.

Break

Keys

Keys and Superkeys of Relations

A set of attributes K in \mathbf{R} is a **superkey** for \mathbf{R} if K functionally determines all attributes of \mathbf{R} ; that is, K uniquely identifies the tuples in \mathbf{R} .

A set of attributes K in \mathbf{R} is a **key** for \mathbf{R} if:

- K is a superkey for \mathbf{R} , and
- no proper subset of K is a superkey.

A proper subset, $K' \subset K$, is a subset of K that does not contain all its attributes, i.e. $K' \neq K$.

Keys and Superkeys of Relations

Example

Imagine a simple database for a library's book inventory:

- Book(Book_ID, Title, Author)

Now,

- {Book_ID}, {Book_ID, Title}, {Book_ID, Title, Author} are *all* superkeys for **Book**.
- {Book_ID} is the only key for **Book**: it is a minimal superkey.

If we also knew that *Title* \rightarrow *Book_ID* *Author* for **Book**, then {Title} would also be a key for **Book**.

Where Do Keys Come From?

You can just assert them, the same way you assert any FD.

Remember: saying that K is a key for \mathbf{R} is the same as saying that

- (a) K functionally determines all attributes of \mathbf{R} , and
- (b) there's no $K' \subset K$ that does so (i.e. K is *minimal*).

Often, you'll be designing \mathbf{R} so it has some sort of "ID" column, perhaps a simple serial number, perhaps a UUID, perhaps some intrinsic unique attribute of \mathbf{R} .

Where Do Keys Come From?

A systematic approach can find all keys, starting from the FDs asserted by the database designer:

1. List the core FDs of the database, and then
2. Deduce the keys by systematic exploration of the **closures** of all possible combinations of attributes.

Definition: *closure* of X in \mathbf{R} is all attributes in \mathbf{R} functionally determined by X in \mathbf{R} .

Closure Test

Say we want to know whether $Y \rightarrow B$ for some \mathbf{R} where we know $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$.

Idea: Test our hypothesis by computing *closure* of attributes determined by Y . Then, see if B is in there.

Closure: algorithm

Write Y^+ to denote "the closure of attributes Y ".

Meaning: Y^+ is all attributes functionally determined by Y , directly and indirectly.

Algorithm: Start from $Y^+ = Y$; then iteratively apply FDs $X_i \rightarrow A_i$ for all $1 \leq i \leq n$ until we reach a fixed point.

Closure: algorithm

Example: Let **R**'s schema be $\{ProductId, ProductName, Category, Department\}$, with FDs

- $ProductId \rightarrow ProductName$
- $ProductId \rightarrow Category$
- $Category \rightarrow Department$

Q. Is it true that $ProductId \rightarrow Department$?

Closure: algorithm

Step	$\{ProductId\}^+$	FD to apply
0	$\{ProductId\}$	$ProductId \rightarrow Category$
1	$\{ProductId, Category\}$	$ProductId \rightarrow ProductName$
2	$\{ProductId, Category, ProductName\}$	$Category \rightarrow Department$
3	$\{ProductId, Category, ProductName, Department\}$	N/A — we are done!

Closure: algorithm

Step	<i>{ProductId}+</i>	FD to apply
0	<i>{ProductId}</i>	<i>ProductId</i> → <i>Category</i>
1	<i>{ProductId, Category}</i>	<i>ProductId</i> → <i>ProductName</i>
2	<i>{ProductId, Category, ProductName}</i>	<i>Category</i> → <i>Department</i>
3	<i>{ProductId, Category, ProductName, Department}</i>	N/A — we are done!

Closure for deducing keys

General procedure:

1. Enumerate possible keys, starting from the smallest.
2. Skip possibilities that are supersets of previously-discovered keys. Otherwise:
3. For each, compute its closure. Is the closure a superkey?
Then we have found a new key!

What is the naive complexity of this algorithm?

Closure for deducing keys

Optimization 1: (Starting.) Are there attributes not mentioned on any FD RHS? They must be in every key!

Optimization 2: (Stopping early.) Would *all* possibilities yet to be examined be supersets of a key we've already found? Then stop early.

Closure for deducing keys

Example: Consider relation **R**(*ABCD*) with FDs:

- $A \rightarrow C$
- $BC \rightarrow D$
- $ABD \rightarrow C$
- $ACD \rightarrow B$

Q. Find all keys of **R**.

Closure for deducing keys

Optimization (Starting): Are there attributes that do not appear in some RHS?

FDs

$A \rightarrow C$

$BC \rightarrow D$

$ABD \rightarrow C$

$ACD \rightarrow B$

Closure for deducing keys

Optimization (Starting): Are there attributes that do not appear in some RHS?

Yes. A is not on any RHS.
 $\therefore A$ must be in every key.
 \therefore don't bother examining keys $\not\supset A$

FDs

$A \rightarrow C$

$BC \rightarrow D$

$ABD \rightarrow C$

$ACD \rightarrow B$

Closure for deducing keys

Examine all 1-element possibilities:

Possibility	Closure	Closure superkey?
$\{A\}^+$	$\{AC\}$	no

FDs

$A \rightarrow C$

$BC \rightarrow D$

$ABD \rightarrow C$

$ACD \rightarrow B$

(remember: we are ignoring possibilities without A)

Closure for deducing keys

Examine all 2-element possibilities:

Possibility	Closure	Closure superkey?
$\{AB\}^+$	$\{ABCD\}$	yes
$\{AC\}^+$	$\{AC\}$	no
$\{AD\}^+$	$\{ABCD\}$	yes

FDs

$A \rightarrow C$

$BC \rightarrow D$

$ABD \rightarrow C$

$ACD \rightarrow B$

(remember: we are ignoring possibilities without A)

Closure for deducing keys

Examine all 3-element possibilities:

Optimization (Stopping): Would every possible key we discover be a superkey of an already-discovered key?

FDs

$A \rightarrow C$

$BC \rightarrow D$

$ABD \rightarrow C$

$ACD \rightarrow B$

Yes in this case. So stop.

Answer: $\{AB\}$ and $\{AD\}$ are the keys of R.

Question

Can we apply the same algorithm to discover all keys of relation Productsales?

**See you at 11:00
tomorrow!**