Introduction to Databases (IDB)

Nathan Sharp (Dr Paolo Guagliardo)

04-12-2020

Contents

IDB Lecture 1: Introduction

Database Management System (DBMS) Advantages

- Uniform data administration
- Efficient access to resources
- Data independence
- Reduced application development time
- Data integrity and security
- Concurrent access
- Recovery from crashes

Database Kinds

- Relational databases (course focus)
- Document stores
- Graph databases
- Key-value stores

Relational Model

First proposed by Edgar F. Codd, 1970

Schema

A relational model has a **schema** consisting of

- set of table names
- column names
- constraints

Query Languages

Procedural Specify a *sequence of steps* to obtain expected results.
Declarative Specify *what* you want not *how* to get it.

- queries are typically declarative (the how is internal).

IDB Lecture 2: Basic Structured Query Language (SQL)

SQL Data Model

- Data is organised in *tables* (aka *relations*)

Tables (Relations) are a collection of *tuples* (aka *rows* or *records*)

SQL

Consists of two sublanguages,
Data Definition Language (DDL): operations on the schema
Data Manipulation Language (DML): operations on the instance

Getting to the UOE psql prompt

Better instructions on pizza:

1. `ssh s1869292@ssh.inf.ed.ac.uk`
2. `ssh student.login`
3. `ssh student.compute` (unnecessary?)
4. `psql -h pgteach`

PostgreSQL (pqsl)

- psql command are case insensitive

Changing the Definition of a Table

```
ALTER TABLE <name>
    RENAME TO <new_name>;
    RENAME <column> TO <new_column>;
    ADD <column> <type>;
    DROP <column>;
    ALTER <column>
        TYPE <type>;
    SET DEFAULT <value>;
    DROP DEFAULT;
```

```
TRUNCATE TABLE <name>;
DROP TABLE <name>;
```

Basic Queries

```
SELECT <list_of_attributes>
FROM <list_of_tables>
WHERE <condtion>
```

- when multiples tables are selected in 'FROM', the tables are concatenated (all rows with all rows/nested loop)

IDB Lecture 3: Basic SQL 2

Database Modification

```
UPDATE <table>
SET <assignments>
WHERE <condition>
```

Joins

Joins are syntactic sugar for filters with multiple tables

```
table1 JOIN table2 ON <condition>
table1 INNER JOIN table2 ON <condition>
table1 LEFT JOIN table2 ON <condition>
table1 RIGHT JOIN table2 ON <condition>
table1 OUTER JOIN table2 ON <condition>
```

Renaming attributes

```
... FROM Customer c, Account [AS] A ...
```

- the `AS` is optional

IDB Lecture 4: Relational Algebra (RA)

Relational Algebra

Relational Algrbra Expression takes an input of relation(s) (R), applies a sequence of operations and returns a relation as an output.

Operations

Projection (π) vertical operation which chooses columns. Of general form

$$\pi_{A_1,\ldots,A_n}(R)$$

taking only the values of attributes A_1 to A_n for each tuple in R.

Selection (σ) horizontal operation on rows. Of general form

$$\sigma_{condition}(R)$$

taking only the tuples in R for which the condition is satisfied.

- for $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_1 \wedge \theta_2}(R)$, the RHS generally has faster runtime.

Product ($\times$) cartesian product *concatenates* each tuple of R with each tuples of S. Of general form

$$R \times S.$$

- conditional on the relations having a disjoint set of atributes
- $cardinality(R \times S) = cardinality(R) \times cardinality(S)$
  - where **Cardinality** is the number of attribues.
- $arity(R \times S) = arity(R) + arity(S)$
  - where **Arity** is the number of rows.

Renaming (ρ) gives a new name to some attribute of a relation with syntax

$$\rho_{replacements}(R)$$

where a replacement has the form $A \rightarrow B$.

Union Intersection & Difference

Note: Relations must have the same attributes.

Union ($\cup$) set of all rows in R and S
Intersection ($\cap$) all rows that belong to both R and S
Difference ($-$) all rows in A that are not in B

Joining relations

Joins can be created by combining Cartesian product ($\times$) with selection (σ).

Natural Join ($\bowtie$) joins two tables on their *common attributes*
Theta-join $R \bowtie_\theta S = \sigma_\theta(R \times S)$
Equijoin $\bowtie_\theta$ where θ is a *conjunction of equalities*
Semijoin $R \ltimes_\theta S = \pi_X(R \bowtie_\theta S)$ where X is the set of attributes of R
Antijoin $R \bar{\ltimes}_\sigma S = R - (R \ltimes_\sigma S)$

Translating SQL to/from Relational Algebra

`SELECT` $\iff$ projection (π)
`FROM` $\iff$ Product ($\times$)
`WHERE` $\iff$ selection (σ)

`SELECT` $A_1, ..., A_n$
`FROM` $T_1, ..., T_m$
`WHERE`
$\updownarrow$
$\pi_{A_1,...,A_n}(\sigma_{<condition>}(T_1 \times ... \times T_m))$
where common attributes in $T_1, ..., T_m$ must be renamed.

IDB Lecture 5: Relational Algebra on Sets

Division

Divison R over a set of attributes X
S over a set of attributes $Y \subset X$
Let $Z = X - Y$
$R \div S = \{r \in \pi_Z(R) \,|\, \forall s \in S, rs \in R\}$
$= \{r \in \pi_Z(R) \,|\, \{r\} \times S \subseteq R\}$
$= \pi_Z(R) - \pi_Z(\pi_Z(R) \times S - R)$

Note: I don't really understand

IDB Lecture 6: Predicate Logic

Free variables variables that are not in the scope of any quantifier. A variable that is not free is bound.

Interpretations

A formula may be true or false w.r.t a given *interpretation*.

Interpretation defines the semantics of the language; an assignment of variables that gives meaning to a statement.

Semantics of FOL: Interpretations

First Order Structure $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$
Δ non empty domain of objects (universe)
$a^{\mathcal{I}}$ function which gives meaning to constant & predicate symbols

- $a^{\mathcal{I}} \in \Delta$ – gives meaning to *constants*, "object a by means of interpretation function $\mathcal{I}$".
- $R^{\mathcal{I}} \subseteq \Delta^1 \times ... \times \Delta^n$ – gives meaning to *predicates*, "mapping it to an element in our domain (objects in the unverse)"

Variable Assignment (v) maps each variable to an object in Δ

- *Notation:* $v[x/d]$ is v with x $\rightarrow$ d

Semantics of FOL: Terms

Interpreatation of terms under $(\mathcal{I}, v)$

$$x^{\mathcal{I},v} = v(x)$$

$$a^{\mathcal{I},v} = a^{x}$$

Formulas

$(\mathcal{I}, v) \models \phi$ means interpretation (I, v) satisfies formula ϕ

$$\begin{aligned}
I, v \models P(t_1, ..., t_n) &\iff (t_1^{\mathcal{I},v}, ..., t_n^{\mathcal{I},v}) \in P \\
I, v \models \neg\phi &\iff \mathcal{I}, v \not\models \phi \\
I, v \models \phi \wedge \psi &\iff \mathcal{I}, v \models \phi \text{ and } \mathcal{I}, v \models \psi \\
I, v \models \phi \vee \psi &\iff \mathcal{I}, v \models \phi \text{ or } \mathcal{I}, v \models \psi \\
I, v \models \phi \rightarrow \psi &\iff \mathcal{I}, v \models \phi \text{ then } \mathcal{I}, v \models \psi \\
I, v \models \forall x\phi &\iff \text{for every } d \in \Delta : \mathcal{I}, v[x/d] \models \phi \\
I, v \models \exists x\phi &\iff \text{there exists } d \in \Delta \text{ s.t } \mathcal{I}, v[x/d] \models \phi
\end{aligned}$$

IDB Lecture 7: Predicate Logic 2

Satisfiability and Validity

A formula is: Satisfiable : if it has a model.

Unsatisfiable if it has no models.
Falsifiable is thre is some interpretation that is not a model.
Valid (tautology) is every interpretation is a model.

Equivalence

Equivalence (≡) Two formulas are *logically equivalent* if they have the same models.

Universal and Existential Quantification

Universal Quantification (∀)

Everyone taking IDS is smart:

$$\forall x(Takes(x, dbs) \rightarrow Smart(x))$$

- typically $\rightarrow$ is the main connective with $\forall$

Existential Quantification ($\exists$)

Someone takes IDS and fails:

$$\exists x(Takes(x, dbs) \wedge Fails(x, dbs))$$

- typically $\wedge$ is the main connective with $\exists$

Quantifier duality

Each quantifier can be expressed using the other.

$$\forall x \text{ Likes}(x, cake) \equiv \neg\exists x \neg \text{ Likes}(x, cake)$$

$$\exists x \text{ Likes}(x, broccoli) \equiv \neg\forall x \neg \text{ Likes}(x, broccoli)$$

Equivalence Properties

Commutativity, Associativity, Distributivity, Idempotence, Absorption, De Morgan, Implication

IDB Lecture 8: Relational Calculus (RC)

- extension of predicate logic

Relational Calculus

A **relational calculus query** is an expression of the form $\{\bar{x} \mid \phi\}$ where,

- head ($\bar{x}$) is a tuple of variables
- body (φ) is a FOL formula
- all the free variables in the body must be mentioned in the head.
- queries without heads are called boolean queries.

Example 1: Name the customers younger than 33 or older than 50 (where Customer = Id, Name, Age).

$$\{y \mid \exists x, z \;\text{ Customer}(x, y, z) \wedge (z < 33 \vee z > 50)\}$$

Example 2: Name and age of customers having an account in London (where Account = Number, Branch, CustID).

$$\{y, z \mid \exists x \;\text{ Customer}(x, y, z) \wedge \exists w \,\text{Account}(w, \text{'London'}, x)\}$$

Example 3: ID of customers who have and account in *every* branch.

$$\{x \mid \exists y, x \text{ Customer}(x, y, z) \wedge (\forall u, w, v \text{ Account}(u, w, v) \rightarrow \exists u' \text{ Account}(u', w, x))\}$$

Interpretations in RC

Every constant is interpreted as itself

Answer to Queries

With every constant fixed, relational calculus queries are really a 'database' as they only operate over the relations.

The answer to a query $Q = \{\bar{x}\varphi\}$ on a database D is

$$Q(D) = \{v(\bar{x}) \mid v : \mathbf{free}(\varphi) \to \delta \text{ such that } D, v \models \varphi\}$$

Safety

Safety A query is safe if it gives a finite answer on all databases and this answer does not depend on the universe Δ.

- Safety Test: can this query give me an infinite answer (on an infinite database)?

IDB Lecture 9: Active Domain & Translating Relational Algebra/Calculus

Active Domain

Active Domain (Adom(R) all constansts occuring in the database (a set of all values).

- Calulating queries with Adom makes for *safe* relational calculus
- queries are finite as there are finitly many elements

Evaluation of Quantifiers under active domain

Assume $\mathbf{Adom}(D) = \{1,2,3\}$

$$\begin{aligned} &D, v \models \exists x R(x, y) \wedge S(x) \\ &\quad\iff \\ &D, v \models (R(1, y) \wedge S(1)) \vee (R(2, y) \wedge S(2)) \vee (R(3, y) \wedge S(4)) \end{aligned}$$

Relational Algebra (RA) ≡ Safe Relational Calculus (RC)

RA and RC are syntactically different but semantically *equally expressive.*

- important as your database engine needs to be able translate your what to a how.

Relational Algebra to Relational Calculus

Translate each RA expression E into a FOL formula φ.

Environment (η) *Injective Map* from attributes to values.

- map convention to be used in class $\eta(A) = x_A$

Base Relation

R over $A_1, ..., A_n$ is translated to $R(\eta(A_1), ..., \eta(A_n))$

Example: If R is a base realtion over A, B

$$\eta = \{A \mapsto x_A, B \mapsto x_B\}$$

Renaming

$$\rho_{\text{OLD}\to\text{NEW}}(E)$$

Process *Rename($\rho_{OLD\to NEW}(E)$) $\to$ RC:*

1. Translate E to φ.
2. If there is no mapping for NEW in η add $\{\text{NEW} \mapsto x_{\text{new}}\}$.
3. Replace every occurrence of *eta*(NEW) in φ with a *fresh* variable.
4. Replace every (free) occurrence of $\eta(\text{OLD})$ in φ by $\eta(\text{NEW})$.

Example: If R is a base relation over A, B then translate the following (RA) to relational calculus, $\rho_{A\to B}(\rho_{B\to C}(R))$.

1. Translate inner bracket $\rho_{B\to C}(R))$

    1. Translate inner bracket R over A, B gives

    $$R(x_A, x_B)$$

    2. No mapping for C (NEW) so adding C to map,

    $$M = \{A \mapsto x_A, B \mapsto x_B, C \mapsto x_C\}$$

    3. no occurrence of x_C (η(NEW)) in $R(x_A, x_B)$ so this step does nothing.
    4. Replacing x_B with x_C gives

    $$R(x_A, x_C)$$

2. Mapping for B so does nothing.
3. No instance of x_B so this step does nothing.
4. Replacing x_A with x_B

$$R(x_B, x_C)$$

Hence,

$$\rho_{A\to B}(\rho_{B\to C}(R)) \iff R(x_B, x_C)$$

Projection

$$\pi(E) \text{ is translated to } \exists X \varphi$$

where

- φ is the translation of E
- $X = \ \mathbf{free}(\varphi) - \eta(\alpha)$
  - attributes that are *not* projected become quantified

Example: For a base relation R over A, B, translate $\pi_A(R)$.

$$\exists x_B R(a_A, x_B)$$

Selection

$$\sigma_\theta(E) \text{ is translated to } \varphi \wedge \eta(\theta)$$

where

- φ is the translation of E
- $\eta(\theta)$ is obtained from θ by replacing each attribute A by $\eta(A)$

Example: For relation R over A, B, translate $\sigma_{A=B \vee B=21}(R)$.

$$R(x_A, x_B) \wedge (x_A = x_B \vee x_B = 21)$$

Product, Union, Difference

Product $E_1 \times E_2$ is translated to $\varphi \wedge \varphi$
Union $E_1 \cup E_2$ is translated to $\varphi \vee \varphi$
Difference $E_1 \times E_2$ is translated to $\varphi \wedge \neg\varphi$

Example: For Relations,

- Customer (C) : CustID, Name
- Account (A) : Number, CustID
- Environment : $\eta = \{\text{CustID} \mapsto x_1, \text{Name} \mapsto x_2, \text{Number} \mapsto x_1\}$

Translate the following, Customer $\bowtie$ Account.

Solution: Expressing the join in primitive operations

$$\pi_{CustID,Name,Number}(\sigma_{CustID=CustID'}(C \times \rho_{CustID \to CustID'}(A)))$$

Translating to RC

1. Customer: $C \Rightarrow C(x_1, x_x)$
2. Account: $A \Rightarrow A(x_3, x_1)$
3. Renaming: $\rho_{CustID \to CustID'}(2) \Rightarrow A(x_3, x_1')$
4. Innermost product: $1 \times 3 \Rightarrow C(x_1, x_2) \wedge A(x_3, x_1)$
5. Selection: $\sigma_{CustID=CustID'}(4) \Rightarrow C(x_1, x_2) \wedge A(x_3, x_1) \wedge x_1 = x_4$

6. Projection: $\pi_{CustID,Name,Number}(5) \Rightarrow$

$$\exists x_4 \, C(x_1, x_2) \wedge A(x_3, x_1) \wedge x_1 = x_4$$

IDB Lecture 10: Translating Relational Calculus to Relational Algebra

Active Domain in Relational Algebra

For a Relation R over attributes $A_1, ..., A_n$

Adom(R) given by $\rho_{A_1 \to A}(\pi_{A_1}(R)) \cup \ldots \cup \rho_{A_n \to A}(\pi_{A_n}(R))$

Adom(D) $\bigcup_{R \in D} Adom(R)$ (where D is a Database)

- *Intuition:* return all elements of database in single column
- we denote $Adom_N$ (where N is a name) to the RA expression that returns the query.

From RC to RA

Translate each FOL formula ρ to a RA expression E

Assumptions

- no universal quantifiers, implications or double negations
  - see L07 "Quantifier Duality" for conversions
- no distinct pair of quantifiers binds the same variable name
- no variable name occurs both free and bound
- no variable names repeated within a predicate
- no constants in predicates
- no atoms of the form x **op** x or c_1 **op** c_2

Predicate

$R(x_1, ..., x_n) \implies \rho_{A_1 \text{is translated to} \eta(x_1), \ldots, A_n \to \eta(x_n)}(R)$

Example: For R over A, B, C, $R(x, y, z)$ is translated to $\rho_{A \to A_x, B \to A_y, C \to A_z}(R)$

Existential Quantification

$\exists x \, \varphi$ is translated to $\pi_{\eta(X - \{x\})}(E)$
where

- E is the translation of φ
- X is free(φ)

Example: For φ with free variables x, y, z and translation E,, $\exists y \varphi$ is translated to $\pi_{A_x, A_z}(E)$.

Comparisons

x **op** y is translated to $\sigma_{\eta(x)\ \mathbf{op}\ \eta(y)}(\mathbf{Adom}_{\eta(x)} \times \mathbf{Adom}_{\eta(y)})$

x **op** c is translated to $\sigma_{\eta(x)\ \mathbf{op}\ c}(\mathbf{Adom}_{\eta(x)})$

- where c is a constant

Example 1: $x = y$ is translated to $\sigma_{A_x=A_y}(\mathbf{Adom}_{A_x} \times \mathbf{Adom}_{A_y})$

Example 2: $x > 1$ is translated to $\sigma_{A_x>1}(\mathbf{Adom}_{A_x})$

Negation

$\neg\varphi$ is translated to $\prod_{x\in\mathbf{free}_\varphi} \mathbf{Adom}_{\eta(x)}) - E$

- where E is the translation of ϕ
- $\prod$ is the Cartesian product

Example: For φ with free variables x, y and translation E, $\neg\varphi$ is translated to $\mathbf{Adom}_{A_x} \times \mathbf{Adom}_{A_y} - E$

Disjunction

$\varphi_1 \vee \varphi_2$ is translated to $E_1 \times (\prod_{x\in X_2-X_1} \mathbf{Adom}_{\eta(x)} \cup E_2 \times (\prod_{x\in X_1-X_2} \mathbf{Adom}_{\eta(x)})$ where

- E_i is the translation of φ_i
- $X_i = \mathbf{free}(\varphi_i)$

Conjunction

Same as disjunction, but uses $\cap$ for $\cup$

Example: For Relations,

- Customer (C) : CustID, Name
- Account (A) : Number, CustID

Translate $\exists x_4 C(x_1, x_2) \wedge A(x_3, x_4) \wedge x_1 = x_4$

Solution:
Environment $\eta = \{x_1 \mapsto A, x_2 \mapsto B, x_3 \mapsto C, x_4 \mapsto D\}$

1. $C(x_1, x_2) \Rightarrow C$
2. $A(x_3, x_4) \Rightarrow \rho_{x_1\to x_4}(A)$
   - to remove clashes
3. $x_1 = x_4 \Rightarrow \sigma_{A=D}(\mathbf{Adom}_A \times \mathbf{Adom}_D)$
4. $2 \wedge 3 \Rightarrow 2 \times \mathbf{Adom}_A \cap 3 \times \mathbf{Adom}_B$
   - note 2 (LHS) and 3 (RHS) do not have same free variables, so we need to add the missing variables to each side

5. $1 \wedge 4 \Rightarrow (C \times \mathbf{Adom}_C \times \mathbf{Adom}_D) \cap 4 \times \mathbf{Adom}_B$
6. $\exists x_4 5 \Rightarrow \pi_{A,B,C}(5)$

Expanding out:

$$\begin{aligned}\pi_{A,B,C}(&(C \times \mathbf{Adom}_C \times \mathbf{Adom}_D)\cap \\ &((\rho_{x_1 \to x_4}(A) \times \mathbf{Adom}_A)\cap \\ &(\sigma_{A=D}(\mathbf{Adom}_A \times \mathbf{Adom}_D)) \times \mathbf{Adom}_B) \times \mathbf{Adom}_B)\end{aligned}$$

- suspicious of final adom?

Equivalent to (solution on slides):

$$\begin{aligned}\pi_{A,B,C}(&(C \times \mathbf{Adom}_C \times \mathbf{Adom}_D) \cap \\ &(A \times \mathbf{Adom}_A \times \mathbf{Adom}_B) \cap \\ &(\sigma_{A=D}(\mathbf{Adom}_A \times \mathbf{Adom}_D)) \times \mathbf{Adom}_B \times \mathbf{Adom}_C))\end{aligned}$$

- Where $C = \rho_{CustID \to A, Name \to B}(C)$
- $A = \rho_{Number \to C, CustID \to D}(A)$

IDB Lecture 11: Multisets and Aggregation

Multisets

Multiset sets where the same element can occur multiples times (SQL uses multisets)
- **Multiplicity** is the number of occurences of an element.
- **Bags** another name for multisets.

Operation to remove multiples (ϵ)

As described.

Basic SQL

```
Q := SELECT [DISTINCT] a FROM t WHERE c
    | Q1 UNION [ALL] Q2
    | Q1 INTERSECT [ALL] Q2
    | Q1 EXCEPT [ALL] Q2
```

`SELECT a FROM t WHERE c` – keeps duplicates, `[DISTINCT]` removes them
`UNION, INTERSECT & EXCEPT` – remove duplicates, `[ALL]` keeps them

SQL to RA on bags

SQL	RA on bags
`SELECT` α ...	$\pi_\alpha(.)$
`SELECT DISTINCT` α ...	$\epsilon(\pi_\alpha(.))$
Q_1 `UNION ALL` Q_2	$Q_1 \cup Q_2$
Q_1 `INTERSECT ALL` Q_2	$Q_1 \cap Q_2$
Q_1 `EXCEPT ALL` Q_2	$Q_1 - Q_2$
Q_1 `UNION` Q_2	$\epsilon(Q_1 \cup Q_2)$
Q_1 `INTERSECT` Q_2	$\epsilon(Q_1 \cap Q_2)$
Q_1 `EXCEPT` Q_2	$\epsilon(Q_1) - Q_2$

- duplicates are good because they give you a true distribution of the data

Aggregate Functions in SQL

`COUNT` number of elements in a column
`AVG` average value of all elements in column
`SUM` Adds up all elements in a column
`MIN / MAX` min/max values of elements in a column
`COUNT (*)` counts all rows in table
`COUNT (DISTINCT *)` is ILLEGAL! use, `SELECT COUNT(DISTINCT T.*)`

IDB Lecture 12: Aggregation with Grouping

For table `Account` with columns `Number, Branch, CustID, Balance, Spend`

1. *How much money does each customer have in total across all of their accounts?*

```
SELECT A.custID, SUM(A.balance)
FROM Account A
GROUP BY A.custID ;
```

2. *How much money is there total in each branch?*

```
SELECT A.branch SUM(A.balance)
FROM Account A
GROUP BY A.branch ;
```

3. *How much money does each customer have in each branch?*

```
SELECT A.custID, A.branch, SUM(A.balance)
FROM Account A
GROUP BY A.custID, A.branch ;
```

4. *Branches with a total balance (across accounts) of at least 500?*

```sql
SELECT A.branch SUM(A.balance)
FROM Account A
GROUP BY A.branch
HAVING SUM(A.balance) >= 500 ;
```

- can't use `WHERE` as its has evaluation precedence over `GROUP BY`

Order of Evaluation Precedence

1. `FROM` – taking rows from (joined) tables listed
2. `WHERE` – discard rows not satisfying the condition
3. `GROUP BY` – partition rows according to attributes
4. **Compute Aggregates**
5. `HAVING` – discard rows not satisfying
6. `SELECT` – output the value of expressions listed
7. *Money available in total to each customer across their accounts?*

```sql
SELECT custID, SUM(A.balance - A.spend)
FROM Account A
GROUP BY A.custID ;
```

IDB Lecture 13: Nested Queries (Subqueries)

Question 1: Accounts with a higher balance that the average of all accounts?

```sql
SELECT A.number
FROM Account A
WHERE A.balance > ( SELECT AVG(A1.balance.)
                    FROM Account A1 ) ;
```

```sql
SELECT ...
FROM ...
WHERE (term_1,...,term_n op (subquery ) ;
```

- valid iff size(subquery) == n

Revisiting `WHERE`

term := attribute | value

Comparison

- `(term,...,term)` **op** `(term,...,term)`
- `term IS [NOT] NULL`
- `(term,...,term)` **op** `ANY (query)`

- `(term,...,term)` **op** `ALL (query)`
- `(term,...,term)` **op** `[NOT] IN (query)`
- `EXISTS (query)`

Condition

- comparison
- condition `AND` condition
- condition `OR` condition
- `NOT` condition

All / Any

- All/Any vs. empty set -> true

IDB Lecture 14: Nested Queries (Subqueries) 2

Question 1. Id of customers from London who own an account?

```sql
SELECT C.ID
FROM Customer C
WHERE C.City = 'London'
  AND C.ID = ANY (  SELECT A.custID
                    FROM Account A ) ;
```

Question 2. Customers living in a city without a branch?

```sql
SELECT *
FROM Customer C
WHERE C.city <> ALL ( SELECT A.branch
                      FROM Account A ) ;
```

- `<> ALL` could be replaced with `NOT IN`

Question 3. Return all the customers if there are some accounts in London?

```sql
SELECT *
FROM Customer C
WHERE EXISTS (  SELECT 1
                FROM Account A
                WHERE A.branch = 'London'
                  AND A.CustID = C.id ) ;
```

Question 4. ID of customers who own an account (living in London)?

```sql
SELECT C.Id
FROM Customer C
WHERE C.City = 'London'
  AND EXISTS (  SELECT *
```

```
                FROM Account A
                WHERE A.CustId = C.id ) ;
```

IDB Lecture 15: Nested Queries (Subqueries) 3

Examples with Exists / Not Exists

- universal quantification expressed with `NOT EXISTS` *Question 1. Customers living in a city without a branch (repeat question)?*

```
SELECT *
FROM Customer C
WHERE NOT EXISTS ( SELECT *
                   FROM Account A
                   WHERE A.brach = c.city ) ;
```

Scoping

A subquery has - a local scope (its `FROM` clause) - an outer scope

Question 2. Branches with a total balance (across accounts) of at least 500?

```
SELECT subquery.branch
FROM ( SELECT A.branch, SUM(A.balance) AS total
       FROM   Account A
       GROUP BY A.branch ) AS subquery
WHERE subquery.total >= 500 ;
```

Question 3: Average the total balances across each customer's accounts?

Strategy 1. find the total balance across each customers accounts 2. take the average of the totals

```
SELECT AVG(subquery.tot)
FROM    ( SELECT   A.custid, SUM(A.balance) AS tot
          FROM     Account A
          GROUP BY A.custid ) AS subquery ;
```

Ordering

Syntax: `ORDER BY` $\langle$column$_1\rangle$ [`DESC`] ,. . . , $\langle$column$_n\rangle$ [`DESC`]

Example 1

```
SELECT *
FROM Accounts
ORDER BY custid ASC, balance DESC;
```

Casting

syntax: `CAST`(term `AS` ⟨ type ⟩)

Conditional Expressions

```
CASE WHEN (bool-exp)
     THEN (value-exp)
     ...
     WHEN (bool-exp)
     THEN (value-exp)
     ELSE (value-exp)
END
```

- ELSE is optional -> NULL if no match

Pattern Matching

Syntax: `term LIKE pattern`

Question 4: Customers with a name that begins with 'K' and has at least 5 characters?

```
SELECT *
FROM Customer
WHERE name LIKE 'K____%' ;
```

IDB Lecture 16: Database Constraints

Integrity constraints

- instances that satisfy the constraints are called **legal**

Functional Dependencies (FD)

Syntax: $X \to Y$, read X *determines* Y

Definition: A relation R satisfies $X \to Y$ if for every two tuples $t_1, t_2 \in R$

$$\pi_X(t_1) = \pi_X(t_2) \implies \pi_Y(t_1) = \pi_Y(t_2)$$

Keys (special case FD)

Definition: A set of attributes X which satisfy

$$\pi_X(t_1) = \pi_X(t_2) \implies t_1 = t_2 \quad \forall t_1, t_2 \in R$$

Intuition: each value in the attribute (column) uniquely identifies the tuple (row)

Inclusion Dependencies (IND)

Syntax: $R[X] \subseteq S[Y]$ where R, S are relations and X, Y are **sequences** of attributes

Definition: R and S satisfy $R[X] \subseteq S[Y]$ if

$$\pi_X(t_1) = \pi_Y(t_2) \quad \forall t_1 \in R \quad \exists t_2 \in S$$

- Note: the projection must respect attribute order

Intuition: the projection of one table must be a subset of a projection of another table

IDB Lecture 17: Database Constraints 2

Not Null

- repetition of stuff I already know

Unique

- allows multiple null

Primary Key

- repetition

Foreign Key

- repetition

IDB Lecture 18: Entailment of Constraints

Implication of Constraints

A set σ of constraints **implies** (or **entails**) a constraint ϕ if *every* instance that satisfies σ also satisfies ϕ

Syntax: $\sigma \models \phi$

Question: Does σ imply ϕ?

Relevance

- do the given constraints imply bad ones
- to the given constraints look bad but imply good ones

Axiomatisation of Constraints

An axiomatisation is–

Sound if every derived constraint is implied
Complete if every implied constraint can be derived

Sound + Complete axiomatisation gives a procedure $\vdash$ such that

$$\sigma \models \phi \iff \sigma \vdash \phi$$

Intuition: if we derive there is an implicit constraint

Armstrong's Axioms (for FDs)

Essential Axioms

Reflexivity $Y \subseteq X \Rightarrow X \to Y$
Argumentation $X \to Y \Rightarrow XZ \to YZ \forall Z$
Transitivity $X \to Y \land Y \to Z \Rightarrow X \to Z$

Derived Axioms

Union $X \to Y \land Y \to Z \Rightarrow X \to YZ$
Decomposition $X \to YZ \Rightarrow X \to Y \land X \to Z$

Closure of a set of FDs

Let F be a set of FDs, the **Closure** (F^+) of F is the set of all FDs implied by the FDs in F.

- can be computed using Armstrong's axioms

Attribute Closure

The **Closure** $(C_f(X))$ **of a set X of Attributes w.r.t. a set F of FDs** is the set of attributes we can derive from X using th FDs in F

$$C_F(X) = \{A \,|\, F \vdash X \to A\}$$

Properties

- $X \subseteq C_F(X)$
- $X \subseteq Y \Rightarrow C_F(X) \subseteq C_F(Y)$
- $C_F(C_F(X)) = C_F(X)$

Solution to implication Problem

$$F \models Y \to Z \iff Z \subseteq C_F(Y)$$

Closure Algorithm

Input: a set F of of FDs and a set X of attributes
Output: $C_F(X)$, the closure of X with respect to F

Algorithm *Closure(F : {FD}, X : {Attribute})* $\rightarrow$ *CFX : {Attribute}*

1. unused := F
2. closure := X
3. **while** $(Y \rightarrow Z) \in$ unused **and** $Y \subseteq$ closure
4. closure := closure $\cup Z$
5. unused := unused $-\{Y \rightarrow Z\}$
6. **return** closure

IDB Lecture 19: Entailment of Constraints 2

Keys, candidate keys and prime candidate

Let R be a relation with set of attributes U and FDs F. $X \in U$ is a **key** for R if $F \models X \rightarrow U$. Equivalently, X is a key if $C_F(X) = U$ as $C_F(X) = U \iff \{A \mid F \models X \rightarrow A\}$.

Candidate Key (minimal set of attributes) X such that $\forall Y \subset X$, Y is not a key.

Prime attribute an attribute of a candidate key.

Computing all Candidate Keys

Algorithm *CandidateKeys(U : {Attribute}, F : {FD})* $\rightarrow$ *CK : {{Attribute}}*

1. $ck := \emptyset$
2. $G(V, E) := V = \{v \mid v \in \mathcal{P}(U)\}$, $E = \{\overrightarrow{XY} \mid X \in v, Y \in V, X - Y = \{A\}\}$
3. **while** G is not empty:
4. $v :=$ node without children
5. **if** $C_F(X) = U$:
6. $ck := ck \cup \{X\}$
7. $G := G - (X + X_{ancestors})$
8. **else**:
9. $G := G - X$

- more optimal variant in the tutorial (lazy expansion of graph)

Implication of Inclusion Dependencies (INDs)

Inclusion Dependency Every X is a Y, such as in a foreign key constraint.
Example: every manager is an employee.

Axiomatisation

Reflexivity $R[X] \subseteq R[X]$

Transitivity $R[X] \subseteq S[Y] \wedge S[Y] \subseteq T[Z] \Rightarrow R[X] \subseteq R[Z]$

Projection $R[X,Y] \subseteq S[W,Z]$ with $|X| = |W| \Rightarrow R[X] \subseteq S[W]$

Permutation $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n] \Rightarrow R[A_{i_1}, ..., A_{i_n}]S[B_{i_1}, ..., B_{i_n}]$ where $i_1, ...i_n$ is a permutation of $1, ..., n$.

FDs and INDs Together

We have shown,

1. Given a set F of FDs and an FD f we can decide whether $F \models f$
2. Given a set G of INDs and an INDs g we can decide whether $G \models g$

Implication Problem: Asking $F \cup G \models f$ or $F \cup G \models g$ is UNDECIDABLE, no algorithm exists can always solve it. This holds for the case of *keys* and *foreign keys* .

IDB Lecture 20: Normal Forms

Redundancy Principle don't repeat constrained information in a table. Every FD should define a key.

Boyce Codd Normal Form (BCNF)

"Problems with bad designs are caused by FDs $X \rightarrow Y$ where X is not a key."

A relation with FDs F is in **BCNF** if $\forall X \rightarrow Y \in F$

1. $Y \subseteq X$ (the FD is trivial), OR
2. X is a key (the closure contains all all attributes in the relation).

- a database is BCNF is all relations are BCNF

Decompositions

Given a set of attributes U and a set of FDs F, a **decomposition** of (U, F) is a set

$$(U_1, F_1), ..., (U_n, F_n)$$

such that $U = \bigcup_{i=1}^{n} U_i$ and F is a set of FDs over U_i

- dont really understand

Criteria for good decompositions

Lossnessness No information is lost

Dependency Preservation no constraints are lost

- formal definitions missed

Projections of FDs

The **projection** of F on $V \subseteq U$ is a subset of the closure containing only the attributes of V.

BCNF Decomposition Algorithm

Algorithm *BCNF-Decomposition(U: {Attribute}, F : {{FD}}) → S : {({Attribute}, {FD})}:*

1. $S := \{(U, F)\}$
2. **while** $\exists (U_i, F_i) \in S$ **not** BCNF:
3. $\quad S[U_i, F_i] :=$ ***decompose(***U_i, F_i)
4. **if** $\exists \{U_i \,|\, (U_i, F_i) \in S, U_i \subseteq U_j, (U_j, F_j) \in S\}$
5. $\quad$ **remove** $S[U_i, F_i]$
6. **return** S

Algorithm *Decompose(*U_i *: {Attribute},* F_i *: {FD})) -> ({U}, {F}):*

1. find $(X \rightarrow Y) \in F$ **not** BCNF
2. V, $Z := C_F(X)$, $U - V$
3. **return** $(V, \pi_V(F))$ **and** $(XZ, \pi_{XZ}(F))$