Intro to DBMS

15 January 2018 20:34

SQL
- NOT NULL
  - must enter data
- NULL
  - optional to add data

- PRIMARY KEY
  - identify entry
- FIREIGN KEY
  - used to link table
    - (no) REFERENCES account
    - no is primary key of account table

Transactions

BEGIN TRANSACTION
- code

COMMIT TRANSACTION

ACID
- Atomicity
  - if crash during code,
    - DBMS can undo incomplete transaction
- Consistency
  - reject transactions that don't keep consistency within the tables
- Isolation
  - can use database as expected while transaction occurring
- Durability
  - once committed, can rely on changes staying there

Shema –structure of the DB
- physical
  - data structures and types
- conceptual
  - logical structure

Relational Model and Algebra

16 January 2018 09:42

Relations take the form $R(A, B, \ldots)$ where
- R is the name of the relation
- $A, B, \ldots$ is the set of attributes of the relation

# of attributes - **arity**

R		
A	B	$\ldots$
v_1^A	v_1^B	$\ldots$
v_2^A	v_2^B	$\ldots$
v_3^A	v_3^B	$\ldots$
$\vdots$		

Set Semantics
- Order of columns not significant
- Order of rows not significant
- No duplicate rows

Column = Attribute
Row = Tuple

Tuples might not contain same amount of entries:
- solution 1: separate relations (tables)
- solution 2: one relation, use NULL for missing entry

Keys:
- every relation has one key which is all the attributes
  - key is violated if two tuples are the same by the extent of the key
- minimal key - no subset of attributes is also a key
- primary key - default key
- foreign key
  - Relation1(Attribute) -> Relation2(Attribute)
  - Left side is a subset of right side attribute

Primitive Operators of the Relational Algebra

Symbol	Name	Type
π	Project	Unary
σ	Select	Unary
$\times$	Cartesian Product	Binary
$\cup$	Union	Binary
$-$	Difference	Binary

Project: like a filter - relation output should not contain any duplicates

Select: evaluate a criteria, If true, return that row - does this for all rows

Product: returns all pairings of tuples (rows)

SPJ:

Union: removes duplicates, merges relations - must be compatible - same number, name and types of columns

Difference: entries in Left hand but not in Right hand - must be union compatible

A monotonic operator has the property that an additional tuple put into any input relation which only cause additional tuples to be generated in the output relation.

A non-monotonic operator has the property that an additional tuple put into an input relation may remove tuples from the output relation

Incremental Query Evaluation
Add extra rows Δ_R to R to become R'

$$R' = R \cup \Delta_R$$
$$\pi_{\vec{X}} R' \equiv \pi_{\vec{X}} R \cup \pi_{\vec{X}} \Delta_R$$
$$\sigma_{P(\vec{X})} R' \equiv \sigma_{P(\vec{X})} R \cup \sigma_{P(\vec{X})} \Delta_R$$
$$R' \times S \equiv (R \times S) \cup (\Delta_R \times S)$$
$$R' \cup S \equiv (R \cup S) \cup \Delta_R$$
$$R' - S \equiv (R - S) \cup (\Delta_R - S)$$
$$S - R' \equiv (S - R) - \Delta_R$$

<u>Derived Operators:</u>

- Natural Join: product of two relationships
  - then select correct pairs -
    - does all columns which are contained in both relations
  - reduce column used for select into one column in new relation

- Semi Join
  - Natural Join of Relation 1 and
    - projection of relation 2, only with columns that appear in both

<u>RA Equivalencies</u>

Project
- eliminate inner project for attributes not used in outer project
- select after project
  - project must contain columns used in select
- Project X of product R S
  - project intersection of X **and** R / S attributes
  - then take product
- Project and union / difference

- Semi Join
  - Natural Join of Relation 1 and
    - projection of relation 2, only with columns that appear in both relations
- Equi join
  - specify which columns to compare
- Theta join
  - select with any function from - cartesian product of two relations
- Foreign Key and Natural Join
  - $A(X) \to FK \to B(X)$
    - # of tuples of join = # of tuples in A

- Intersection
  - $R \cap S = R - (R - S)$
  - find ones that don't match - then remove from R
    - **appear in both tables**

- Division **- use to group / sort**
  - $R \div S$
  - Find in R, value which has entry for all attribute X, where S has all of X listed

- Project X of product R S
  - project intersection of X **and** R / S attributes
  - then take product
- Project and union / difference
  - project R and S first
  - then tale union / difference

Select
- if select after project - can do select before
- two select - simplify to one
- Select from product
  - select **from one** relation
  - then product with other
- select from union
  - select from each relation
  - then take union
- select from difference
  - select on one relation
  - then take difference

Ones we can expand:
  Product of Union
  Product of Difference

Difference and Union
  $R - (S \cup T) = (R - S) - T$

Datalog

26 January 2018 14:24

Predicate: one for each row - same amount of arguments as attributes in relation

- name starts with lower case
- intentional - rule
    - head :- Body
    - head has one predicate
    - body can be any conjunction of predicates
- extensional - data

Variables

- name start with upper case
- if only appear once - replace with '_'

Minimal Model????

Negation:

- block out some results
- must appear in a predicate before
- to negate - use ¬ before name

RA in datalog

- Projection :

  π_{sortcode} account

  account_sortcode(Sortcode) :-
  account(_, _, _, _, Sortcode).

- Select

  $\sigma_{\text{amount}>1000}$ movement

  big_credit(Mid, No, Amount, Date) :-
  movement(Mid, No, Amount, Date),
  Amount $>$ 1000.

- Product

  branch $\times$ $\sigma_{\text{rate}>0}$account

  product_example(BSortcode, BName, Cash, No, Type, CName, Rate, ASortcode) :-
  branch(BSortcode, BName, Cash),
  account(No, Type, CName, Rate, ASortcode),
  Rate $>$ 0.

- Join

$\pi_{\text{bname,cname}}\,\sigma_{\text{branch.sortcode}=\text{account.sortcode}}(\text{branch} \times \text{account})$

```
branch_customers(BName, CName) :-
        branch(BSortcode, BName, _),
        account(_, _, CName, _, ASortcode),
        BSortcode = ASortcode.
≡
branch_customers(BName, CName) :-
        branch(Sortcode, BName, _),
        account(_, _, CName, _, Sortcode).
```

- Union

$\sigma_{\text{amount}>1000}\,\text{movement} \cup \sigma_{\text{amount}<-100}\,\text{movement}$

```
big_movement(Mid, No, Amount, Date) :-
        movement(Mid, No, Amount, Date),
        Amount > 1000.
big_movement(Mid, No, Amount, Date) :-
        movement(Mid, No, Amount, Date),
        Amount < −100.
```

- Difference

$\pi_{\text{no}}\,\text{account} - \pi_{\text{no}}\,\text{movement}$

```
dormant_account(No) :-
        account(No, _, _, _, _),
        ¬movement(_, No, _, _).
```

SQL

30 January 2018 09:46

Structured Query Language

- Data Definition Language / Data Manipulation Language
  - CREATE TABLE table_name
    (attribute_name data_type NULL/NOT NULL
    CONSTRAINT table_name_pk PRIMARY KEY (attribute)
    CONSTRAINT table_name_fk FOREIGN KEY (attribute)
    REFRENCES other_table
)
  - data types

Some SQL Data Types	
Keyword	Semantics
BOOLEAN	A logical value (TRUE, FALSE, or UNKNOWN)
BIT	1 bit integer (0, 1, or NULL)
INTEGER	32 bit integer
BIGINT	64 bit integer
FLOAT(n)	An n bit mantisa floating point number
REAL	32 bit floating point number (≅ FLOAT(24))
DOUBLE PRECISION	64 bit floating point number (≅ FLOAT(53))
DECIMAL(p,s)	A p digit number with s digits after the decimal point
CHAR(n)	A fixed length string of n characters
VARCHAR(n)	A varying length string of upto n characters
DATE	A calendar date (day, month and year)
TIME	A time of day (seconds, minutes, hours)
TIMESTAMP	time and day together
ARRAY	An ordered list of a certain datatype
MULTISET	A bag (i.e. unordered list) of a certain datatype
XML	XML text

  - secondary / candidate keys

Declaring Primary Keys after table creation

```
ALTER TABLE branch
ADD CONSTRAINT branch_pk PRIMARY KEY (sortcode);
```

Declaring Secondary Keys for a table

```
CREATE UNIQUE INDEX branch_bname_key ON branch(bname)
```

  - Insert
    INSERT INTO table_name
    VALUES (tuple),
    (tuple)
  - Update
    UPDATE table_name
    SET attribute = value
    WHERE attribute = value
  - Delete
    DELETE
    FROM table_name
    WHERE attribute = value

Rough implementation of RA:
SELECT attribute_names
from table_name
e.g. branch.bname
branch.* - all columns
FROM table_names
WHERE attribute = value
AND extra attribute = value

e.g.

$\pi_{\text{bname,no}}\,\sigma_{\text{branch.sortcode=account.sortcode} \wedge \text{account.type='current'}}(\text{branch} \times \text{account})$

```
SELECT branch.bname,
       account.no
FROM   account, branch
WHERE  account.sortcode=branch.sortcode
AND    account.type='current'
```

Binary operators between SELECT statements

- SQL UNION implements RA $\cup$
- SQL EXCEPT implements RA $-$
- SQL INTERSECT implements RA $\cap$

Note that two tables must be **union compatible**: have the same number and type of columns

SQL doesn't care about column name - does it via column position

SQL Joins

Modern SQL Join Syntax

```
SELECT branch.*, no, type, cname, rate
FROM   branch JOIN account ON branch.sortcode=account.sortcode
```

Special Syntax for Natural Join

```
SELECT *
FROM   branch NATURAL JOIN account
```

Another Special Syntax for Natural Join

```
SELECT branch.*, no, type, cname, rate
FROM   branch JOIN account USING (sortcode)
```

SELECT DISTINCT
removes duplicates
required when attribute is not a key

Defaults Set / Bag
SELECT ALL
UNION DISTINCT
EXCEPT DISTINCT
INTERSECT DISTINCT
FROM / WHERE have no DISTINCT

RA and SQL	
RA Operator	SQL Operator
π	SELECT
σ	WHERE
$R_1 \times R_2$	FROM R_1, R_2 *or* FROM R_1 CROSS JOIN R_2
$R_1 \bowtie R_2$	FROM R_1 NATURAL JOIN R_2
$R_1 \overset{\theta}{\bowtie} R_2$	FROM R_1 JOIN R_2 ON θ
$R_1 - R_2$	R_1 EXCEPT R_2
$R_1 \cup R_2$	R_1 UNION R_2
$R_1 \cap R_2$	R_1 INTERSECT R_2

Set Operators:

IN

- test for membership of a set
- can use SELECT to generate the set to test against

```
SELECT no
FROM   account
WHERE  type='current'
AND    no IN (SELECT no
              FROM   movement
              WHERE  amount>500)
```

- can place with join
  - can't replace NOT IN with join

EXISTS

- test if a select statement returns any rows
- EXCEPT can be replaced with NOT EXISTS in some cases

ALL and SOME

- test if a value is equal to ALL or SOME values in a relation
- SOME - at least 1
- **returns names of branches** that only have current

names of branches that only have current accounts

```
SELECT bname
FROM   branch
WHERE  'current'=ALL (SELECT type
                      FROM   account
                      WHERE  branch.sortcode=account.sortcode)
```

Null:

- process WHERE
  - true, false or unknown
- Null is usually close to false option
- when evaluating WHERE rate = NULL
  - return nothing - can't know if any value == NULL
- To check if a null value is stored
  - use: WHERE x IS/IS NOT NULL

AND

P_1 AND P_2		P_2: TRUE	UNKNOWN	FALSE
P_1	TRUE	TRUE	UNKNOWN	FALSE
	UNKNOWN	UNKNOWN	UNKNOWN	FALSE
	FALSE	FALSE	FALSE	FALSE

OR

P_1 OR P_2		P_2: TRUE	UNKNOWN	FALSE
P_1	TRUE	TRUE	TRUE	TRUE
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
	FALSE	TRUE	UNKNOWN	FALSE

NOT

		NOT P_1
P_1	TRUE	FALSE
	UNKNOWN	UNKNOWN
	FALSE	TRUE

IS NOT TRUE != IS FALSE
== EITHER FALSE OR UNKNOWN

EXCEPT:

- will treat one null same as another null when comparing

SET based - gets rid of duplicate nulls - doesn't matter how many
BAG based - one null only gets rid of ONLY one null

SQL Programming

06 February 2018 10:08

Pattern Matching

Testing Strings against a Pattern

```
WHERE column LIKE pattern ESCAPE escape_char
```

Will return TRUE where pattern matches column. The escape_char may be used before any of the special characters below to allow them to be treated as normal text.

- _ to match a single character
- % to match any number (including zero) of characters

<u>Modifying Data:</u>

DONE in the SELECT clause

Need to AS after the function

e.g. ABS(), ROUND,UPPER()..

COALESCE(column, value) - turn null into value

CASE:

inside of SELECT

- CASE
  - WHEN
  - THEN
  - ELSE
  - END AS cloumn name
- e.g.

```
SELECT no,
       COALESCE(rate,0.00) AS rate,
       CASE
       WHEN rate>0 AND rate<5.5
       THEN 'low rate'
       WHEN rate>=5.5
       THEN 'high rate'
       ELSE 'zero rate'
       END AS interest_class
```

Relationally Complete SQL

Fully implement RA

TEMPORARY

create temporary tables

- stays throughout session

New table in FROM

(SELECT FROM) name

- free to optimise

<u>Left and Right JOIN</u>

- If row doesn't match, add but with null to fill the tuple
- Left looks at missing form left table, vice-verse

Outer Join

- union of left and right JOIN
- NATURAL FULL OUTER JOIN
  - will COALESCE same columns

<u>OLTP and OLAP</u>

Online Transactional / Analytical Processing

OLTP - read write to a few rows

OLAP - read many rows - analysis

<u>OLAP</u>

- **GROUB BY**
  - one row output per group
  - put's all NULL's into one group
  - aggregate functions applied to non-grouped columns

Aggregate	Semantics
SUM	Sum the values of all rows in the group
COUNT	Count the number of non-NULL rows in the group
AVG	Average of the non-NULL values in the group
MIN	Minimum value in the group
MAX	Maximum value in the group

    - do these functions in SELECT section
    - COUNT needs DISTINT or ALL
    - Null doesn't contribute to aggregate functions
  - **HAVING**
    - add filter to groups
      SELECT
      FROM
      GROUP BY
      HAVING
    - can use to avoid dodgy maths
- **OVER (PARTITION BY)**
  - inside SELECT section
    - doesn't change amount or rows in output
      - but does change ordering of rows
  - internal groups - to use aggregate functions

RANK () OVER (ORDER BY) AS rank

- calculate a rank for a row

PIVOT

COUNT(CASE)

UNPIVOT

```
SELECT no,
       'cname' AS col,
       cname AS value
FROM   account
UNION
SELECT no,
       'type',
       type
```

Order of execution

FROM
WHERE
GROUP BY
HAVING
SELECT
ORDER BY

```
SELECT no,
       'cname' AS col,
       cname AS value
FROM   account
UNION
SELECT no,
       'type',
       type
FROM   account
UNION
SELECT no,
       'rate',
       CAST(rate AS VARCHAR)
FROM   account
WHERE  rate IS NOT NULL
UNION
SELECT no,
       'sortcode',
       CAST(sortcode AS VARCHAR)
FROM   account
```

SELECT
ORDER BY

Functions

CREATE FUNCTIONS

Entity Relationship Modelling

16 February 2018 13:31

ER Schema
Entity - boxes - sets of things - nouns
Relationship - lines connecting boxes -verbs - bi-directional
Attributes - circles from an Entity
- end in ? - nullable
- underline - key

Look Here: - entity next to the constraint
L:U
at least L times
at least U times
U = N - no limit

Look Across
L...U

Subset entity:
specialised - noun
Arrow to Super set entity

Constructs

Construct	Description
$\mathcal{C}$	Look-across cardinality constraints
$\mathcal{L}$	Look-here cardinality constraints
$\mathcal{K}$	Key attributes
$\mathcal{M}$	Mandatory attributes
$\mathcal{O}$	Optional attributes
$\mathcal{S}$	Isa hierarchy between entities

Disjoint Subsets: - D
Superset is a generalisation
Middle box-thing

Weak Entity: - W
- cannot exist without another entity is it related to
- gets its primary key from that other entity
- double box
- underline Cardinality Constraint

Hyper-edge: - H
- relationships between more than two entities
- inability to express constraints with LH when H - but can with Look Across

Attributes: - A
- allow attributes on relationships

Multi-Value: -V
- attributes can have more than one value - ? * +

Nested: - N
- nested relationships

Extended ER

Construct	Description
$\mathcal{A}$	Attributes can be placed on relationships
$\mathcal{D}$	Disjointness between sub-classes can be denoted
$\mathcal{C}$	Look-across cardinality constraints
$\mathcal{H}$	hyper-edges (n-ary relationships) allowed
$\mathcal{L}$	Look-here cardinality constraints
$\mathcal{K}$	Key attributes
$\mathcal{M}$	Mandatory attributes
$\mathcal{N}$	Nested relationships
$\mathcal{O}$	Optional attributes
$\mathcal{S}$	Isa hierarchy between entities
$\mathcal{V}$	Multi-valued attributes
$\mathcal{W}$	Weak entities can be identified

Table per Type

table_name(column_name*)

One-many Relationships

- column in table one added
- is a foreign key to table two
- 0 to many - optional key

Many-many Relationships

- Table of Relationship with the initial primary keys
- each column of new table is foreign key to tables of entities

Subset:

- key of superset becomes key of subset table
- this column is primary key for each table,
  - but is also foreign key between the two tables

Mapping to a relational model

- D - treat disjoint subset as just subsets
- W - weak entity uses a compound key of its primary and the other entity primary key
- H - rules of binary relationship extend to n-ary
- A - attributes go into table as columns
- N - map inner R as normal
  - mapping outer R - treat inner R as an Entity
- V - entity_multi-valued-attribute as new table

Functional Dependencies

23 February 2018 14:09

FD - values of attribute X agree in two tuples, then they must agree in attribute Y for the same two tuples
$X \rightarrow Y$
X implies Y
Y DOES NOT imply X

$XY\ shorthand\ for\ X \cup Y$

Armstrong's Axioms
- Reflexivity - if Y is a subset of X , $X \rightarrow Y$
- Augmentation - if $X \rightarrow Y$, $XZ \rightarrow YZ$
- Transitivity - if $X \rightarrow Y and\ Y \rightarrow Z$ then $X \rightarrow Z$

Implied rules:
- Union Rule - if $X \rightarrow Y and\ X \rightarrow Z$ then $X \rightarrow YZ$
- Pseudotransitivity Rule - if $X \rightarrow Y and\ WY \rightarrow Z$ then $WX \rightarrow Z$
- Decomposition Rule - if $X \rightarrow Y and\ Z \subseteq Y$ then $X \rightarrow Z$

FD and keys:
- Super key X - set of attributes X functionally determines all other attributes
- Minimal key - Super key which you cannot remove any more attributes from

Closure of attributes:
- Start with $X^+ \rightarrow X$
- **apply each FD to the above**

Closure of FD set:
- two sets equivalent if their closures are equivalent

- to close a set - **expand to get all the implied FD's**

Minimal Cover S_C:

- cannot remove any FD's - with the set still having the same closure
- like the **opposite of closure - minimise / 'factorise' the set**

Algorithm:

- Re-write each FD to FD's with only one attribute on the RHS

- consider redundancy in each FD's LHS
  - if $X \to A, and \quad B \in X\ and\ X \to B$
  - replace with $(X - B) \to A$

- For each FD $X \to A$, compute X^+ without using $X \to A$
  - if $A \in X^+$, remove $X \to A$ from set

OR

- Spot transitivity and remove as required

Normalisation

04 March 2018 21:53

Normal Form - **<u>1NF</u>**

- every attribute depends on a key

<u>Prime Attribute:</u>

- where X is a minimal candidate key
- any attribute A is prime if $A \in X$

- to check, find all minimal keys - then see if A is an element

3rd Normal Form - **<u>3NF</u>**

- non-key attribute depends only on the key
- Every non-trivial FD $X \rightarrow A$ either:
  - X is a super-key - determines all other attributes in R
  - A is prime

- to check if a decomposition is 3NF
  - apply above rules to each relation

2. For each FD $X \rightarrow Y$ in $\mathcal{F}_c$, create a relation with schema XY
3. Eliminate a relation if its schema is a subset of another
4. If none of the schemas created so far contains a key of R, add a relation schema containing a key of R

Boyce-Codd Normal Form - **<u>BCNF:</u>**

- every FD $X \rightarrow A$, X is a super-key

Sometimes, decomposing 3NF to BCNF will lose FDs

Based on a BCNF violation $X \rightarrow Y$, decompose R into two relations:

- One with $X \cup Y$ as its attributes
  (i.e., everything in the FD)
- One with $X \cup (attrs(R) - X - Y)$ as its attributes
  (i.e., left side of FD plus everything not in the FD)

Lossless Decomposition:

- union of attributes of new relations = set of attributes of original R
- Natural Join of new relations = original Relation

To produce:

  use FD's to extract a new relation from main

    repeat - decreasing main relation - see Worksheet 13 Q2.

    want to preserve as many FD's as possible when splitting

To check if not LD:

- check if the attribute used to join two relations
  - is in the LHS of an FD

Generating 3NF:

- Check which FD's violate 3NF
- Decompose on those FD's and repeat

Preserving FDs during decomposition:

- Project closure of FD set onto decomposed relations
- the union of these subsets = closure of original FD set
  - hence FDs preserved

Concurrency

06 March 2018 09:20

ACID:

- Atomicity
- Consistency
- Isolation
- Durability

BEGIN TRANSACTION
…(read and write internally)
COMMIT TRANSACTION

Object: O_j - initials_of_table$_{key}$

Commit - c_i - make update available
Abort - a_i - discard

Concurrent Transaction:

- pipeline execution
- must have same result as if executing them in serial order - cannot change order within an instruction
- don't commit if there is uncommitted writes on same data by another transaction

Lost update - replace a write with NOP - and get same end result - BAD - not serializable - is recoverable

Inconsistent Analysis - read only instructions - but reading **before** another transaction writes to same object(s)

Dirty Reads - serializable - read on uncommitted data - problem IF previous transaction aborts - not recoverable if commit of previous transaction is after commit of current

Dirty Write - write to same object before other transaction commits

Lost Update	**Write to NOP**	not S	is R
Inconsistent A	**Read** -> Write		
Dirty Read	Write -> **Read**		RC, if C2 after C1
Dirty Write	Write -> **Write**		

Recoverability:

- RC - **don't commit earlier if dirty reads**
- ACA - **don't dirty read**
- ST - **don't dirty read or dirty write**

ST in ACA in RC

Checking if concurrent:

- only consider committed projections
- check if H_{mixed} is equivalent to any serial order of T's
- conflicts:
  - two transactions do more than just both read on same object
  - serializable (CSR) - if set of conflicts ordered in same serial order
- **serialisation graph**
  - node for each transaction in H
  - edge represents a conflict on an object between the two Transactions
  - H is CSR is if graph is acyclic

Maintaining Serialisability and Recoverability:

- two-phase locking
    - read or write lock on objects
    - lock, read/write, unlock
- refuse read lock if
    - write lock on object by another transaction
- refuse write lock if
    - read or write lock on object by another transaction
- if lock refused, delay transaction from running
- two phases:
    - growing phase
    - shrinking phase
    - **so release only after all gain locks**

Scheduler:

- **Aggressive**
    - gain lock just before
    - unlock at end of transaction
        - Strict locking:
            - MUST release write lock at end
        - STRONG Strict
            - MUST release both read and write at end
            - all at same time
    - maximises concurrency, might suffer from delays or even deadlocks

Waits-for-graph:

- nodes are transactions
- arrow
    - source: transaction waiting
    - dest: transaction being waited on
    - label: lock causing the wait
- deadlock when:
    - cycle from arrows between transactions

- Conservative
    - gain all ASAP
    - unlock when?
    - remove risks of delays later on, might refuse to start
        - prevents deadlocks
        - not recoverable - can abort after releasing write lock

Recovery

04 April 2018 03:07

Log Disc:

- record of changes - used by recovery manager in case of failure
- deal with system failures

REDO:

- must write a REDO if:
  - committed transaction not on disk
- must write to log before commit

UNDO:

- must write an UNDO if:
  - non-committed transaction is on disk
- must write UNDO to log before writing to data disk
- don't need to UNDO on aborts

A LOG

Must contain

- REDO information for each update
- UNDO information for each update
- commit of each transaction

Might contain

- begin of each transaction
  - can be inferred from first REDO/UNDO
  - presence useful to stop search of UNDO records
- abort of each transaction
  - can be inferred from lack of commit
  - presence useful to indicate UNDO already done

Basic Recovery Procedure:

1. Scan back through the log
   a. collect set of committed transactions, C
   b. collect set of incomplete transactions, I
2. Scan back through the log
   - perform UNDO far any transaction in I
3. Scan forward through the log
   - perform REDO for any transaction in C

Omitting the REDO log - requires flushing committed data to disk, not mem:

- collect set of committed transactions, C
- any objects changed in C, put in set D
- back through the log, perform UNDO for objects not in D
- for objects in D, perform UNDO if after
- + no after images are needed
- - high I/O

Omitting the UNDO log - never write to disk uncommitted data:

- must never write uncommitted data to disk
- add fix command to stop Cache manager flushing data
- flush or unfix data after commit
- + no before images needed

Omitting Both

- atomic commit - out pf place updating

Checkpointing:

- saving state mid history
  - quicker to recover
  - limits size of log

Commit Consistent Checkpoint:

- stop accepting new transactions - finish existing ones
- flush all dirty cache to disk
- write CP to log - recover to this point
  - possible long hold-up at CP creation

Cache Consistent Checkpoint:

- suspend all transactions
- flush all dirty cache to disk
- write list of suspended transactions to log
- write CP to log
- recovery:
  - scan back through log to get C and I up until CP
  - perform UNDOs if I before after CP
  - perform UNDOs of incomplete from list of suspended transactions
  - perform REDOs of C

Adjustment - Fuzzy Checkpointing:

- like cache but,
- flush dirty data from cache that wasn't flushed in previous CP
- in recovery, use penultimate CP as target state

Media Failures:

- RAID-1 to have active clone disk for both log and data
- RAID-1 log, have active and archive data disk
  - active periodically updates the archive
  - use logs to restore active form archive

- hard backups
  - requires a CP to be made
  - dump log - records since last log dump
  - dump data - entire database data
  - recover - apply saved log to saved data

SQL as RA implementation

07 February 2018 21:20

person table:

- name - primary key

person						
name	gender	dob	dod?	father?	mother?	born_in
Alice	F	1885-02-25	1969-12-05	null	null	Windsor
Andrew	M	1960-02-19	null	Philip	Elizabeth II	London
Andrew of Greece	M	1882-02-02	1944-12-03	George I of Greece	null	Athens
Anne (Princess)	F	1950-08-15	null	Philip	Elizabeth II	London
Charles	M	1948-11-14	null	Philip	Elizabeth II	London
⋮						

person(father) $\overset{fk}{\Rightarrow}$ person(name)   person(mother) $\overset{fk}{\Rightarrow}$ person(name)

monarch Table:

monarch			
name	house?	accession	coronation?
James I	Stuart	1603-03-24	1603-07-25
Charles I	Stuart	1625-03-27	1626-02-02
Oliver Cromwell	null	1649-01-30	null
Richard Cromwell	null	1658-09-03	null
Charles II	Stuart	1659-05-25	1626-02-02
James II	Stuart	1685-02-06	1685-04-23

monarch(name) $\overset{fk}{\Rightarrow}$ person(name)

prime_minister table:

prime_minister		
name	party	entry
David Cameron	Conservative	2010-05-11
Gordon Brown	Labour	2007-06-27
Tony Blair	Labour	1997-05-02
John Major	Conservative	1990-11-28
Margaret Thatcher	Conservative	1979-05-04
James Callaghan	Labour	1976-04-05
Harold Wilson	Labour	1974-03-04
Edward Heath	Conservative	1970-06-19
⋮		

prime_minister(name) $\overset{fk}{\Rightarrow}$ person(name)

Q1.
Write an SQL query that returns the scheme (name,father,mother)
ordered by name containing the name of all people known to have died before both their father and mother,
together with the name of the mother and the name of the father.

Q2.
Write an SQL query returning the scheme (name)
ordered by name that
lists all people that have either been:
a King, Queen or Prime Minister.

Q3.
A King or Queen is said to abdicate if their **reign ceases before their death.**
Write an SQL query returning the scheme (name)
ordered by name that
lists the name of all Kings or Queens that have abdicated

Q4.
Write a query that returns the scheme (house,name,accession)
ordered by accession that
lists house and name of
monarchs who were the first of a house to accede to the throne.
Maximum marks will be given only to answers that use either the ALL or SOME operators.

SQL as programming language

07 February 2018 23:38

person table:

- name - primary key

person						
name	gender	dob	dod?	father?	mother?	born_in
Alice	F	1885-02-25	1969-12-05	null	null	Windsor
Andrew	M	1960-02-19	null	Philip	Elizabeth II	London
Andrew of Greece	M	1882-02-02	1944-12-03	George I of Greece	null	Athens
Anne (Princess)	F	1950-08-15	null	Philip	Elizabeth II	London
Charles	M	1948-11-14	null	Philip	Elizabeth II	London
⋮						

person(father) $\overset{fk}{\Rightarrow}$ person(name)   person(mother) $\overset{fk}{\Rightarrow}$ person(name)

monarch Table:

monarch			
name	house?	accession	coronation?
James I	Stuart	1603-03-24	1603-07-25
Charles I	Stuart	1625-03-27	1626-02-02
Oliver Cromwell	null	1649-01-30	null
Richard Cromwell	null	1658-09-03	null
Charles II	Stuart	1659-05-25	1626-02-02
James II	Stuart	1685-02-06	1685-04-23

monarch(name) $\overset{fk}{\Rightarrow}$ person(name)

prime_minister table:

prime_minister		
name	party	entry
David Cameron	Conservative	2010-05-11
Gordon Brown	Labour	2007-06-27
Tony Blair	Labour	1997-05-02
John Major	Conservative	1990-11-28
Margaret Thatcher	Conservative	1979-05-04
James Callaghan	Labour	1976-04-05
Harold Wilson	Labour	1974-03-04
Edward Heath	Conservative	1970-06-19
⋮		

prime_minister(name) $\overset{fk}{\Rightarrow}$ person(name)

Q5.
Write an SQL query that returns the scheme (first name,popularity)
ordered in descending order of popularity,
and then alphabetical order of first name.
Your answer should also
exclude first names that only occur once in the database.
A first name is taken to mean the first word appearing the name column of person.

Q6.
Write an SQL query that returns the scheme
(house,seventeenth,eighteenth,nineteenth,twentieth)
ordered by house
listing the number of monarchs of each royal house that acceded to the
throne in the 17th, 18th, 19th and 20th centuries.

Q7.
Write an SQL query returning the scheme (father,child,born)
ordered by father,born that
lists as father the name of all men in the database,
together with the name of each child,
with born being the number of the child of the father (i.e. returning 1 for the first born, 2 for
the second born, etc).
For men with no children, the man should be listed with null for both
child and born.

Q8.
Write an SQL query that returns the scheme (monarch,prime minister),
ordered by monarch and prime minister,
that lists prime ministers that held office during the reign of the monarch.

ER

18 February 2018 18:27

You are to design a new database to hold data on various species of animals.

Each animal **species** will be **identified by its name**, and we will record the **average weight**, **length** and **lifespan** of the species. If a species is **extinct, we record the date of the extinction**. Where known, we also record for a given species the other species from which the given **species evolved.**

For all the **countries of the world** we record the country **name**, **land area** of the country, and the **name of the main organisation for wildlife monitoring** in the country. For all **continents** we record the continent **name**, and the **land area of the continent.**

We record the **population** of a species that is estimated to **live in** the land area of each country that falls within each continent (hence, for example, recording separate populations of species that estimated to live in European Russia and Asian Russia).

The species may be divided into reptiles, insects, **fish, birds or mammals.** However our database will only store information on the last three.

For **fish**, we will record the number of **gills** and the number of **fins** the species has, and the **names of all the seas** in which the fish is found.

For **birds**, we will record the **average wing span**, and if the bird **flies or not.**

For **mammals** we record the number of **legs** and the number of **cortical areas** found in the brain. If the mammal has a tail, then we record the **length of the tail**. Some mammals may be classed as placental mammals, and for those placental mammals, we will record the **gestation period** of offspring, and the **size of the placenta**.

animal_species(name, average_weight, length, lifespan, extinct?)

animal_species_evolved_from(name, evolved_from)
animal_species_evolved_from(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

fish(name, gills, fins)
fish(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

fish_seas(name, seas)
fish_seas(name) $\overset{fk}{\Rightarrow}$ fish(name)
fish_seas(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

birds(name, average_wing_span, flies)
birds(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

mammals(name, legs, cortical_areas, tail_length?)
mammals(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

placental(name, gestation_period, placenta_size)
placental(name) $\overset{fk}{\Rightarrow}$ mammals(name)
placental(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

continent(name, land_area)

country(continent.name, country.name, land_area, monitoring_organisation)
country(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)

land_area(continent.name, country.name)
land_area(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)
land_area(country.name) $\overset{fk}{\Rightarrow}$ country(country.name)

lives_in(animal_species.name, continent.name, country.name)
lives_in(animal_species.name) $\overset{fk}{\Rightarrow}$ animal_species(name)
lives_in(continent.name, country.name) $\overset{fk}{\Rightarrow}$ land_area(continent.name, country.name)
lives_in(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)
lives_in(country.name) $\overset{fk}{\Rightarrow}$ country(country.name)

Relational schema

animal_species(name, average_weight, length, lifespan, extinct?)

animal_species_evolved_from(name, evolved_from)
animal_species_evolved_from(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

fish(name, gills, fins)
fish(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

fish_seas(name, seas)
fish_seas(name) $\overset{fk}{\Rightarrow}$ fish(name)
fish_seas(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

birds(name, average_wing_span, flies)
birds(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

mammals(name, legs, cortical_areas, tail_length?)
mammals(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

placental(name, gestation_period, placenta_size)
placental(name) $\overset{fk}{\Rightarrow}$ mammals(name)
placental(name) $\overset{fk}{\Rightarrow}$ animal_species(name)

continent(name, land_area)

country(continent.name, country.name, land_area, monitoring_organisation)
country(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)

land_area(continent.name, country.name)
land_area(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)
land_area(country.name) $\overset{fk}{\Rightarrow}$ country(country.name)

lives_in(animal_species.name, continent.name, country.name)
lives_in(animal_species.name) $\overset{fk}{\Rightarrow}$ animal_species(name)
lives_in(continent.name, country.name) $\overset{fk}{\Rightarrow}$ land_area(continent.name, country.name)
lives_in(continent.name) $\overset{fk}{\Rightarrow}$ continent(name)
lives_in(country.name) $\overset{fk}{\Rightarrow}$ country(country.name)