Introduction to Database

Transactions

ACID Properties

Atomicity	all or nothing
Consistency	consistent before and after
Isolation	independent of any other transaction
Durability	completed transaction are durable

Examples for ACID properties:

Atomicity

When system crashes while a transaction is performing, the change will be undone on next restart.

Consistency

Changes must be made to all connected tables;
No duplicated primary keys.

Isolation

When a cash transfer is in progress, just after the amount is removed from the sender's account, system sums the cash in the bank.
Isolation ensures the result to be correct.

Durability

System informed the user that the cash transfer has been done, but it crashes while a transaction is performing.
On next restart, system must continue to process the half transaction to ensure the information given to user is correct.

(Or system can inform the user only if the transaction is really done).

The Relational Data Model

Relations

Relations take the form $R(A, B, \ldots)$ where

- R - name of the relation
- A, B - set of attributes
  - A, B, C can also be written as AB
  - arity - number of attributes n of the relation
  - $Domain(A)$ - set of values that the attribute can have
  - $Attrs(R)$ returns ‘AB‘
- extent of $R(AB)$ is set of tuples $\{\langle v_1^A, v_1^B \rangle, \langle v_2^A, v_2^B \rangle, \langle v_3^A, v_3^B \rangle, \ldots\}$

Example:

account		
no	type	sortcode
100	‘current’	67
101	‘deposit’	67
103	‘current’	34
107	‘current’	56
119	‘deposit’	56
125	‘current’	56

- *Relation*, R: the table, e.g. ‘account’
- *Attributes*: the columns ‘no’, ‘type’ and ‘sortcode’
- *Arity* = number of attributes = 3
- *Tuple*: the pair of the 3 values in each row, e.g. $(100, ‘current', 67)$
- $Domain(type) = \{‘current', ‘deposit'\}$
- $Domain(no) = \{100, 101, 103, 107, 119, 125\}$
- $Domain(sortcode) = \{67, 34, 56\}$
- $Attrs(R) = \{no, type, sortcode\}$

Keys

A key of a relation R(AB) is a subset of the attributes for which the values in any extent are unique across all tuples.

- violated key - there being two tuples in the extent which have the same values for the attributes of the key. (i.e. duplicated)
- minimal key - set of attributes AB ...for which no subset of the attributes is also a key
- primary key - the default key when no key explicitly stated
- foreign key - $R(\vec{x}) \overset{fk}{\Rightarrow} S(\vec{y})$ y is a key of S

$$account(sortcode) \overset{fk}{\Rightarrow} branch(sortcode)$$

account		
no	type	sortcode
100	'current'	67
101	'deposit'	67
103	'current'	34
107	'current'	56
119	'deposit'	56
125	'current'	56

branch	
sortcode	cash
56	132456
34	97836
67	45000

sortcode is a foreign key.

The Relational Algebra

Primitive operators of the Relational Algebra

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

Project π

Project is just like slicing the table.

Project returns sets which contain no duplicates.

project 是管理列的，select 管理行的。

$\pi_{sortcode\ as\ id}$ can change the name of the attribute.

account		
no	type	sortcode
100	'current'	67
101	'deposit'	67
103	'current'	34
107	'current'	56
119	'deposit'	56
125	'current'	56

$\pi_{no,type}$ **account**	
no	type
100	'current'
101	'deposit'
103	'current'
107	'current'
119	'deposit'
125	'current'

$\pi_{sortcodeasid}$ **account**
id
56
34
67

Select σ

Select is like 'filter'.

Note that the selection expression must be valid.

account			
no	type	rate	sortcode
100	'current'	NULL	67
101	'deposit'	5.25	67
103	'current'	NULL	34
107	'current'	NULL	56
119	'deposit'	5.50	56
125	'current'	NULL	56

$\sigma_{rate>0}$ **account**			
no	type	rate	sortcode
101	'deposit'	5.25	67
119	'deposit'	5.50	56

Multiple conditions can be written with methematical logical operators:

$$\sigma_{branch.sortcode=account.sortcode \wedge account.type=`current'}$$

Product $\times$

Multiplies two tables. Disambiguation is done by specifying 'table.attr'.

A 3-row table $\times$ a 2-row table gives a result table of 6 rows.

account	
no	sortcode
100	67
101	67
103	34

bccount	
no	type
101	'deposit'
119	'current'

account × bccount			
account.no	sortcode	bccount.no	type
100	67	101	'deposit'
100	67	119	'current'
101	67	101	'deposit'
101	67	119	'current'
103	34	101	'deposit'
103	34	119	'current'

Union ∪

Merging two tables with exactly same attributes, which is called *union compatible*.

account	
no	sortcode
100	67
101	67
103	34

bccount	
no	sortcode
104	77
119	95

account ∪ bccount	
no	sortcode
100	67
101	67
103	34
104	77
119	95

Difference −

Differing two tables with exactly same attributes.

account
no
100
101
103

bccount
no
100
101

account − bccount
no
103

Intersection ∩

Find the common part of the two tables.

Division ÷

$$D = Attr(R) - Attr(S)$$

$$R \div S = \pi_D R - \pi_D \left(\pi_D R \times S\right) - R$$

[redacted]

Derived Operators

Natural Join $\bowtie$

Natural Join is a combination of Cartesian Product and Select of their common primary key, as expressed:

$$R \bowtie S = \sigma_{R.A_1=S.A_1 \wedge \ldots \wedge R.A_m=S.A_m} R \times S$$

Example:

$$branch \ \bowtie \ account = \sigma_{branch.sortcode=account.sortcode} branch \times account$$

So there won't be two attributes 'sortcode' from both of the tables.

Example: Add the **branch** to **account** with same sortcode, as shown:

account		
no	name	sortcode
100	Alice	67
101	Bob	67
103	Catherine	34

branch	
sortcode	cash
67	97340
34	8900

account $\bowtie$ branch			
no	name	sortcode	cash
100	Alice	67	97340
101	Bob	67	97340
103	Catherine	34	8900

Semi Join

$$R \ltimes S = R \bowtie \pi_{Attr(R) \cap Attr(S)}(S)$$

Example:

$$account \ltimes movement = account \bowtie \pi_{no}(movement)$$

Theta Join

$$R \overset{\theta}{\bowtie} S = \sigma_\theta R \times S$$

Equi Join

$$R \overset{A=B}{\bowtie} S = \sigma_{R.A=S.B} R \times S$$

SQL Data Definition Language

Definition of Tables

```
CREATE TABLE branch(
    sortcode INTEGER NOT NULL,
    bname VARCHAR(20) NOT NULL,
    cash DECIMAL(10, 2) NOT NULL
);
CREATE TABLE account(
    no INTEGER NOT NULL,
    type VARCHAR(8) NOT NULL,
    cname VARCHAR(20) NOT NULL,
    rate DECIMAL(4, 2) NULL,
    sortcode INTEGER NOT NULL,
    CONSTRAINT account_pk PRIMARY KEY (no),
    CONSTRAINT account_fk FOREIGN KEY (sortcode) REFERENCES branch
)
```

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);
```

SQL Data Manipulation Language

RA Operator	SQL Expression
π	SELECT DISTINCT
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

Mixed examples:

To list people with a current and a deposit account,

```
SELECT current_account.cname,
       current_account.no AS current_no,
       deposit_account.no AS deposit_no
FROM   account AS current_account
       JOIN account AS deposit_account
       ON current_account.cname=deposit_account.cname
       AND current_account.type='current'
       AND deposit_account.type='deposit';
```

Below are more detailed examples for each operators.

Changing Data

```
INSERT INTO account
VALUES (100, 'current', 'McBrien, P.', NULL, 67),
        (101, 'deposit', 'McBrien, P.', 5.25, 67);
```

```
UPDATE account
SET type='deposit'
WHERE no=100;
```

```
DELETE
FROM account
WHERE no=100;
```

Selecting Data

Corresponds to RA Projection (π).

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

RA π is distinct while SQL SELECT is not.

$\pi_A B =$ 'SELECT DISTINCT A FROM B;'. When attribute A is the primary key, 'DISTINCT' can be omitted since its already distinct.

```
SELECT no FROM account;
SELECT DISTINCT type FROM account;
```

Binary operators between SELECT statements

UNION RA $\cup$

EXCEPT RA $-$

INTERSECT RA $\cap$

Note that tables must be *union compatible*.

```
SELECT no FROM account
EXCEPT
SELECT no FROM movement;
```

SQL Join

CLASSIC JOIN

```
SELECT branch.*, no, type
FROM branch, account
WHERE branch.sortcode = account.sortcode;
```

Modern JOIN

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

Special Natural Join

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

Another Special Natural Join

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

Note that tables must be *union compatible*.

SQL Set Operations

Simple filtering

```
SELECT *
FROM account
WHERE type='current'
AND no IN (100, 101)
```

This ‘SELECT’ returns ‘no = 100’ or ‘no = 101’ only.

Negating IN

```
SELECT *
FROM account
WHERE type='current'
AND no NOT IN (100, 101)
```

Nested selection

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

≡

```
SELECT DISTINCT account.no
FROM account JOIN movement
    ON account.no=movement.no
WHERE type='current'
AND amount>500
```

‘

Selection EXISTS

```
SELECT DISTINCT cname
FROM account
WHERE cname NOT IN(
  SELECT cname
  FROM account
  WHERE type = 'deposit'
)
```

≡

```
SELECT DISTINCT cname
FROM account
WHERE NOT EXISTS(
  SELECT *
  FROM account AS a2
  WHERE type='deposit'
  AND account.cname=cname
)
```

‘

Correlated Subquery

A correlated subquery contains a reference to the columns of the outer query.

Result is as if the subquery were executed for each row considered by the ‘WHERE’ clause.

```
SELECT DISTINCT cname
FROM account AS a1
WHERE NOT EXISTS(
  SELECT *
  FROM account AS a2
  WHERE type='deposit'
  AND a1.cname=a2.cname
)
```

SQL SOME and ALL

SOME returns true if there is at least one matching.

ALL returns true if all values match.

names of branches that only have current accounts

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

names of branches that have deposite accounts

```
SELECT bname
FROM branch
WHERE 'deposit' = SOME(
 SELECT type FROM account WHERE branch.sortcode=account.sortcode
)
```

SQL Nulls

SQL implements three valued logic

[redacted]

Comparing NULL in SQL Queries

‘WHERE rate = NULL’ and ‘WHERE rate <> NULL’ are wrong.

Use ‘WHERE rate IS NULL’ and ‘WHERE rate IS NOT NULL’ instead.

Testing for logical truth value

```
SELECT no
FROM account
WHERE (rate=5050) IS NOT TRUE
```

NULL means Might Be

Comparing NULL with a exact value yields a UNKNOWN, which means ‘Might Be’.

In ‘WHERE’ clauses, ‘rate=5.25’ accepts the NULL rates.

‘WHERE (rate=5.25) IS NOT FALSE’ allows Might Be while ‘WHERE (rate=5.25) IS TRUE’ or simply ‘WHERE (rate=5.25)’ don’t.

Equivalences Between EXCEPT, NOT IN and NOT EXISTS

[redacted]